LAZARUS FREE PASCAL

Développement rapide

Matthieu GIROUX

Programmation

Livre de coaching créatif par les solutions

ISBN 9791092732214
et 9782953125177

Éditions LIBERLOG
Éditeur n° 978-2-9531251

Dépôt Légal RENNES 2010

Sommaire

Élever à la dignité d'homme tous les individus de l'espèce humaine

Lazare Carnot

À LIRE

Les mots que vous ne comprenez pas, avec leur première lettre en majuscule, peuvent être trouvés dans le glossaire à la fin du livre.
Les exemples peuvent se télécharger sur www.liberlog.fr/ftp.

Les mots en gras indiquent des chapitres.
Les mots en majuscules indiquent des groupements, des langages, des outils, des entités, des marques.
Les mots entre guillemets indiquent des mots de LAZARUS, des mots à chercher sur le Web, ou des mots de votre environnement.
Les mots entre crochets indiquent des mots à remplacer.

OBJECTIFS DU LIVRE

L'objectif du livre est d'apprendre facilement LAZARUS afin de créer rapidement un Logiciel. On met en avant la création de Composants, étape indispensable dans l'automatisation et l'optimisation. On donne la démarche pour créer très rapidement un Logiciel. Cette démarche consiste à éviter les copiés-collés pour créer des exécutables binaires sur toutes les plates-formes ou du Byte Code sur Android.

Le Développement Rapide d'Applications est mis en avant dans ce livre. Le Développement Très Rapide d'Applications est l'aboutissement de tout savoir-faire automatisant la création de Logiciel. On prépare la création pour sauter une étape : La création du Logiciel à partir de l'analyse. C'est l'analyse qui crée le Logiciel grâce à votre moteur DTRA.

Le livre a aussi pour objectif de promouvoir des sources qui sont parmi les meilleures sources laissées temporairement de côté. Le moteur de jeux ZEN GL et les serveurs web EXTPASCAL sont parmi les meilleurs savoir-faire de LAZARUS.

LICENCE

DU MÊME AUTEUR

L'Économie pour les Petits
L'Économie pour les Enfants
Pourquoi y aurait-il un Dieu ?
Expliquer sa religion chrétienne
Devenir un Génie
Favoriser la Créativité
L'Économie est Physique.
La généalogie c'est gratuit, avec les logiciels libres
France – Fonctionnement de notre société
Les Deux France
LAZARUS et le Développement Rapide
Comment Écrire des Histoires
L'astucieux LINUX
Nos Nouvelles Nos Vies
Poèmes et Sketchs – De 2003 à 2008

Visibles chez thebookedition.com et archive.org.

À editions.liberlog.fr, comment-ecrire.fr, devenircreatif.com, quiestdieu.net, lazarus-components.org, economie-reelle.org, economieenfants.com, economiepetits.com, france-analyse.com, etrecatholique.com, moralemetiers.com, programmationinfos.com.

Du même éditeur

Alors vous voulez tout savoir sur l'économie ?
Lyndon Larouche
Mon Père m'a Dit
Elliott Roosevelt
Les Principes de la Science Sociale
Henry Charles Carey

Biographie

Matthieu GIROUX aime l'écriture, avec la recherche de la vérité par le dialogue, utilisant des moments de réflexion, permettant de se souvenir et de se construire.

Ayant écrit des poèmes puis des nouvelles, il a écrit un livre sur l'écriture. La démarche consistait a écrire pour les débutants, car celui qui apprend ou réapprend est lors de cet apprentissage le plus accessible aux débutants. Ce premier livre a permis de revoir la pédagogie d'un journal d'un réseau international.

Il s'intéresse au Développement Rapide d'Applications, afin de trouver de meilleures techniques de programmation. Sauter une étape de conception pour la création de Logiciel est possible grâce à l'automatisation, maître mot de l'informatique dans l'entreprise.

LAZARUS peut être une étape dans la réalisation d'un savoir-faire ou Framework. Un Framework sert à gagner un maximum de temps. Il faudra ensuite automatiser à partir d'autres Frameworks afin de mieux faire connaître le sien. Recherchez le partage pour vous enrichir et enrichir les autres.

Ayant vu qu'il ne suffisait pas de gagner du temps, il parle d'économie de travail dans un livre sur l'économie. Cela permet de montrer comment on crée nos libertés et nos droits, grâce à l'économie physique.

Devenu auto-éditeur, s'intéresser à la politique permet d'être encore plus convaincant en visant juste. Matthieu GIROUX a publié des livres phares sur

l'économie physique, comme la méthode Larouche Riemann, ou des livres de Henry Charles Carey et d'Emmerich de Vatel. Il écrit des livres sur l'économie physique, sur la France, sur la créativité.

LAZARUS FREE PASCAL

CREATIVE COMMON BY SA

HISTOIRE

Lazarus était un disciple du Christ. Jésus Christ ressuscita Lazarus. Depuis Lazarus est associé à la légende du Phénix. Le titre LAZARUS FREE PASCAL vient des chercheurs français Blaise PASCAL puis Lazare CARNOT. Lazare est écrit Lazarus aux États-Unis. Lazare CARNOT a organisé la victoire de Napoléon, afin de créer des écoles prestigieuses en France, notamment Polytechnique et l'École Normale, le CNAM. Il a été associé à Gaspard MONGE et sa pédagogie.

Sadi CARNOT est dit Nicolas Léonard Sadi CARNOT aux États-Unis. C'est le fils de Lazare. Sadi CARNOT, physicien-chimiste du XIXe, fut le surdoué de polytechnique. Il élabora réellement la thermodynamique. Le concept physique de dynamique, au lieu d'effet, venait par ailleurs de LEIBNIZ au début du XVIIIe. LEIBNIZ dialogua avec Denis PAPIN pour sa première machine à vapeur. Denis PAPIN a aussi inventé la roue à aube, reprise par Benjamin FRANKLIN au XIXe.

Sadi CARNOT a dû partir aux États-Unis, à cause de concurrences dans l'armée napoléonienne. Les inventions toujours fonctionnelles de Denis PAPIN sont passées de la même manière aux oubliettes, à tel point qu'en France peu connaissent en 2014 Denis PAPIN et le travail des CARNOT. Le physicien-chimiste Sadi CARNOT laisse un travail achevé sur la thermodynamique, avec son seul ouvrage publié nommé "Réflexions sur la puissance des machines à feu". Ce livre définit l'ensemble des axiomes de la thermodynamique. Les autres livres ont été brûlés.

Nicolas Léonard Sadi CARNOT était passionné par la transformation du corps et de l'esprit. Mes livres de coaching créatif et compréhensif par les solutions sur la créativité et sur l'écriture vous permettent de commencer cette transformation.

Ce fut ensuite Vladimir VERNADSKI qui impacta la physique quantique.

La communauté Lazarienne est décrite par la communauté Sainte-Rosalie. Sadi CARNOT le physicien mourut dans une contagion aux États-Unis, pour que son livre soit réutilisé par Kelvin 35 ans plus tard. Kelvin était complaisant avec les doctrines libérales et malthusiennes.

Vidéos gratuites :
Sadi Carnot par Michel SERRES (de La Légende des Sciences)
Moteur à vapeur de Denis PAPIN à Blois et en Bretagne
Livre Libre sur la thermodynamique chez www.framabook.org.
Livre "Réflexions sur la puissance des machines à feu" sur www.wikisource.fr.

POURQUOI LE LANGAGE PASCAL ?

Le PASCAL est un langage des plus simples et des plus ingénieux. Il permet de trouver facilement les erreurs. Le Compilateur FREE PASCAL permet de créer des exécutables indépendants de librairies complémentaires. Vous pouvez ainsi avec plus facilement diffuser vos Logiciels.
LAZARUS est basé sur un langage nommé FREE PASCAL. FREE PASCAL est un Compilateur PASCAL Orienté Objet écrit à l'origine en C. L'assembleur, le langage le plus proche de la machine, s'ajoute cependant au FREE PASCAL, notamment pour WINDOWS et LINUX.
Le langage FREE PASCAL possède une grammaire et des instructions PASCAL, basées sur l'algorithmique. C'est le langage permettant d'apprendre la programmation. Il suffit de traduire les mots anglais du PASCAL vers le français pour retrouver le langage de l'algorithmique. Le PASCAL est donc enseigné pour cette raison particulière.

L'algorithmique du PASCAL permet de facilement trouver les erreurs, afin de se concentrer sur l'évolution de son intuition ou recherche de vérité, pour évoluer plus facilement en précisant alors sa démarche. On peut alors envisager l'ingénierie et ses enjeux de gain de temps grâce à la simplicité.
Sinon le langage PASCAL permet, une fois acquis, de jouer facilement sur l'utilisation des mots. on simplifie alors sa compréhension de la science. En plus

du latin, ce langage permet de communiquer facilement dans son domaine scientifique, avec les informaticiens aussi.

POURQUOI CHOISIR LAZARUS ?

Si vous voulez apprendre la programmation d'interfaces homme-machine LAZARUS est fait pour vous. LAZARUS c'est un EDI Libre de DRA. Un EDI est un Environnement de Développement Intégré. Un EDI sert à faciliter la programmation. Le DRA est le Développement Rapide d'Applications. Le DRA permet de créer vite et bien les programmes. LAZARUS est multi-plates-formes, disponible donc sur un maximum de machines.

Le projet LAZARUS a commencé en 1999. Il est actuellement préférable d'utiliser LAZARUS plutôt que KYLIX ou DELPHI XE8. KYLIX est un DELPHI sous LINUX créé en 2001. KYLIX a été abandonné dès sa troisième version parce qu'il était propriétaire dans le monde du Logiciel Libre.

DELPHI est beaucoup enseigné dans les écoles françaises. Il est aussi utilisé par les industries et par les éditeurs, parce qu'il est efficace. LAZARUS ne possède pas de licence payante. Mais son atout décisif en 2015 est d'être présent sur les deux bureaux majeurs disponibles sur GNU LINUX ainsi que sur WINDOWS et les deux bureaux MAC OSX.

Vous pouvez adapter des Composants DELPHI pour que vos Logiciels deviennent multi-plates-formes. Ils seront donc utilisables sur beaucoup de machines. DELPHI a été créé en 1995 par BORLAND, avant LAZARUS en 1999. LAZARUS a bénéficié du partage de l'équipe BORLAND.

LAZARUS est compatible avec le compilateur de DELPHI à 99 %. Cependant le verbiage récent du compilateur DELPHI a créé certains Frameworks PASCAL Objet inspirés du verbiage de la programmation JAVA, incompatible avec LAZARUS. JAVA et FREE PASCAL permettent cependant de simplifier la sémantique des listes. Les listes PASCAL devenues typées s'inspirent de JAVA.
LAZARUS c'est une utilisation des librairies Libres avec une interface facile à utiliser, l'algorithmique PASCAL. LAZARUS est un Logiciel Libre et gratuit,

fonctionnant sur le partage. Le partage permet la sûreté d'un Logiciel. Les deux licences libres utilisées sont la LGPL et la GPL. Participer à LAZARUS nécessite de partager vos Sources.
LAZARUS c'est le Développement Rapide d'Applications par les licences libres. Une Source non partagée dort. Ainsi, 80 % des Sources des Logiciels Propriétaires sont Libres.

Un Composant est une partie réutilisée et centralisée de votre Logiciel. La gestion des Composants a été améliorée sur LAZARUS. Les problèmes connus de gestion de Composants sur DELPHI ont été réglés. LAZARUS est fait pour gagner un maximum de temps dans la création de Logiciels complexes ou simples, avec les Composants, la complétion, le travail du code automatisé, etc. Il peut aussi être utilisé pour apprendre facilement la programmation (cf fin du chapitre **Programmer facilement**).

Un Logiciel Client/Serveur c'est en quelque sorte un Logiciel qui n'a pas besoin de navigateur web pour fonctionner. Si vous voulez créer ce genre de Logiciel, vous pouvez compter sur des librairies de Composants visuels, permettant de réaliser simplement vos interfaces exécutables. Ces exécutables sont directement compris par la machine. LAZARUS permet de créer les interfaces des mobiles et des ordinateurs.

Si vous voulez créer votre jeu multi-plates-formes rapidement, LAZARUS possède des librairies permettant de créer des jeux sur 2 Dimensions ou sur 3 Dimensions. On utilise LAZARUS pour créer des Logiciels graphiques multi-plates-formes économiques. Il existe des librairies permettant de lire des formats de fichiers 2D ou 3D, Libres ou propriétaires, pour WINDOWS, LINUX, MAC OSX, iOS, ANDROID. Des librairies 3D Libres nécessitent continuellement la participation.

Les Frameworks LAZARUS ont certes peu de concurrence, mais ils sont convaincants dès leur début. Les équipes LAZARUS FREE PASCAL et anciennement BORLAND, recherchent la simplicité. Cela fait que les Frameworks de DRA ont peu besoin de simplifier leur utilisation. Les premiers Frameworks PASCAL Objet sont très souvent les bons.

L'atout principal du PASCAL c'est l'algorithmique avec les Composants de DRA. Cet ensemble permet d'évoluer rapidement selon son intuition. On devient alors un très bon ingénieur Objet. L'atout de LAZARUS c'est l'association par les licences Libres et le partage, venant notamment de l'équipe BORLAND et des indépendants.

POURQUOI LE DRA EST-IL IMPORTANT ?

La nécessité de l'utilisation du Développement Rapide d'Applications est simple et large. Notre économie est basée historiquement et scientifiquement sur l'économie de travail. Le DRA simplifie la programmation pour accélérer la création logicielle, afin de créer notamment des applications LAZARUS rapides et intuitives, diffusées facilement parce que Libres, ou bien parce que basées sur des Sources Libres.

La création de l'ergonomie sur un outil de DRA se fait de deux manières : Par la souris accompagnée du clavier pour débuter ou pour personnaliser. Puis les Composants de DRA sont améliorés pour être automatisés selon l'économie de travail.

On peut alors facilement améliorer ce que l'on a programmé, grâce à l'architecture, avec la construction, en décrivant des bâtiments par exemple, selon la simplicité de la pédagogie par l'écriture notamment (cf livres **Comment Écrire des Histoires**, **Favoriser la Créativité**).

ARCHITECTURES FREE PASCAL

Avec LAZARUS, en 2017, vous pouvez créer des Interfaces Homme-Machine fonctionnant sous LINUX KDE et GNOME, WINDOWS, MAC OSX, iOS, ANDROID, WIN CE, WII.

Vos exécutables peuvent aussi fonctionner sous les systèmes BSD, UNIX, OSX, sur certains Systèmes de processeurs spécifiques aussi. Pour certaines plates-

formes il est nécessaire de participer au projet LAZARUS.

Les programmes utilisent actuellement une architecture 32 bits ou 64 bits. On peut schématiser cela comme 32 ou 64 voies d'autoroutes, permettant donc d'accélérer les calculs.

Ainsi une adresse 32 bits peut adresser jusqu'à 2^{32} entiers de 32 bits. Une adresse 64 bits peut adresser jusqu'à 2^{64} entiers de 64 bits. Une adresse 64 bits ne fonctionne pas sur une architecture 32 bits. Par contre une adresse 32 bits peut fonctionner sur une architecture 64 bits, dans la limite de ses possibilités.

LAZARUS fonctionne sur les architectures 32 et 64 bits. Cependant vous devez vérifier que les Composants complémentaires fonctionnent en 64 bits. En effet, en 2013, les serveurs fonctionnent en général en 64 bits. C'est cependant toujours l'architecture 32 bits pour laquelle s'inspire l'architecture 64 bits.

Applications Libres LAZARUS

Une Application Open Source n'est que partagée. Une Application Libre peut s'utiliser, s'étudier, se distribuer et se modifier grâce au partage.

Il existe des applications multi-plates-formes faites avec LAZARUS. LAZARUS est choisi pour le multi-plates-formes Libre.

ANCESTROMANIA permet de créer votre généalogie. Avec, on peut imprimer beaucoup de documents de suivi. Il est possible d'exporter un GEDCOM, pour www.geneanet.org aussi, pour son propre site web, etc.

LAZPAINT est un des projets LAZARUS des plus prolifiques. Il dispose en effet d'une librairie complète de dessin, de Composants graphiques et un peu plus...

SKYCHART et VIRTUAL MOON sont des applications scientifiques Libres en 3 dimensions. Il y a aussi un Logiciel libre de retouches photos pour

l'astronomie. Vous pouvez visualiser vos scans IRM grâce à des Logiciels libres créés avec LAZARUS.

Différents Logiciels de gestion vous permettent de gérer votre café, entreprise ou restaurant. Vous pouvez créer rapidement un Logiciel de gestion avec les savoirs-faire EXTENDED, MAN FRAMES et XML FRAMES, nécessitant une participation, comme tout savoir-faire de DRA.

PEA ZIP est un compresseur Libre concurrent de 7 ZIP, permettant donc de regrouper des fichiers afin qu'ils prennent moins de place.

Il existe des applications Libres permettant de gérer les données FIREBIRD, PARADOX, ORACLE, MY SQL, MARIA DB, ou SQLITE.

ORTHONET est un Logiciel éducatif Libre. Des professeurs ont choisi LAZARUS. Il existe aussi différents utilitaires Libres pour la vidéo ou le son.

Des sites Web vous permettent de télécharger pour participer à ces Logiciels Libres. Vous avez deux listes de Logiciels utilisant LAZARUS avec ces liens :
http://wiki.lazarus-ide/Projects_using_LAZARUS/fr

Du PASCAL orienté Objet

L'Orienté Objet signifie que l'on utilise l'Objet pour ce qu'il a de plus utile et efficace. La programmation par Objets est la programmation la plus proche de l'humain. Cette programmation regroupe des parties du programme dans des Objets. On peut ainsi facilement représenter son programme en entités.

Comme tout outil Objet de DRA, LAZARUS est accessible. Il peut cependant vous emmener, petit à petit, vers les Composants. Vous apprenez alors réellement l'Objet, ses capacités et repères surtout, permettant d'améliorer ce que vous utilisez.

Les entités ou Objets du PASCAL Orienté Objet ont un comportement, une

hiérarchie, des valeurs. Le PASCAL Objet possède des Objets pouvant être enregistrés comme Composants. Les Composants PASCAL améliorent le Polymorphisme du PASCAL Orienté Objet, cette capacité des Objets à accepter plusieurs formes.

LAZARUS ne fait que reprendre les instructions existantes du PASCAL Orienté Objet, grâce au Compilateur multi-plates-formes FREE PASCAL, point de départ de LAZARUS. Le Compilateur FREE PASCAL évolue maintenant grâce à LAZARUS.

Nous vous apprenons à connaître les instructions permettant de réaliser votre Logiciel adapté à vos besoins. Pour aller plus loin, connaître l'Objet et le détail des différentes architectures logicielles permet d'améliorer LAZARUS.

LA COMMUNAUTÉ

LAZARUS dispose d'une communauté active. Cette dernière ajoute au fur et à mesure de nouvelles possibilités. Des communautés LAZARUS s'ouvrent régulièrement dans le monde. En 2015, suite au manque de sûreté de WINDOWS, beaucoup d'entreprises ont migré vers LINUX dans le monde. Ainsi LAZARUS et les paquets EXTENDED ont été téléchargés et choisi en 2015. Des savoir-faire payants se sont alors portés vers LAZARUS.

Avec le site web www.freepascal.org, Il existe un wiki LAZARUS, ainsi qu'un espace de documentation français. Un espace de traduction facilite la compréhension de LAZARUS.

Il y a aussi deux Roadmap en anglais, l'une qui résume sur www.lazarus-ide.org, mais aussi une autre participative sur bugs.freepascal.org,. Vous pouvez participer à ces espaces, mais surtout donner vos avis sur le forum LAZARUS.

LAZARUS EST PARTAGÉ

Une œuvre est Libre si on peut l'exécuter, la comprendre, la redistribuer, la modifier librement. Contrairement aux Logiciels propriétaires, où on achète sans savoir ce qui est induit dans l'achat, le Logiciel Libre permet la participation, afin que le Logiciel évolue par le développeur en fonction de la demande utilisateur. Les Licences Libres servent à pérenniser les acquis, pour les partager et les redistribuer au maximum de pays.

LAZARUS est un Logiciel Libre. Vous pouvez donc intégrer en théorie le projet LAZARUS, après avoir suffisamment compris l'humain, puis l'ingénierie des Composants, ou celle d'un nouvel environnement ou un nouveau type de bureau, voire du langage assembleur, de sa plate-forme.

Les sites comme www.sourceforge.net, www.bitbucket.com et www.github.com diffusent des Logiciels Open Source. SOURCEFORGE, BITBUCKET et GITHUB permettent en effet de partager vos Sources de Logiciel pour être vu et référencé. Alors d'autres développeurs peuvent améliorer votre Logiciel, car les Sources sont la recette de votre Logiciel.

Sourceforge est le site web qui contient les versions les plus à jour de LAZARUS. Sourceforge permet de disposer d'une mise à jour des Sources. Ces fichiers Sources n'ont pour certaines pas passé les tests.

LES VERSIONS DE LAZARUS

On a intérêt à mettre à jour LAZARUS, afin de disposer des dernières améliorations Libres. Les développeurs LAZARUS gèrent des numéros de version.

Le Versioning LAZARUS se présente avec des chiffres, disposés de cette façon :
A.B.CC-D

Quand vous voyez RC dans la version, il s'agit d'une version de tests servant à rapporter des bugs sur la page d'annonce de la Release Candidate. Après avoir vérifié la date de la RC, il faut alors aller sur le forum pour voir les erreurs ou en

rajouter.

LAZARUS évolue continuellement. Évitez certaines versions.
Le premier nombre nommé "A" présente une version majeure de LAZARUS. Par exemple la version 1.0.00-0 de LAZARUS permet de traduire entièrement tout Composant DELPHI 6 vers LAZARUS. Plus ce chiffre ou nombre est grand et récent, plus cette version est recommandée.

Le deuxième chiffre ou nombre présente une version mineure de LAZARUS. Plus ce nombre ou chiffre est grand, plus cette version est recommandée. Si le deuxième chiffre est impair il s'agit d'une version de transition. Le troisième ou quatrième chiffre sera alors impair.

Le troisième nombre présente la maturité de la version mineure. Si le nombre est pair, la version est stable et utilisable, car elle a été testée.

Si le troisième nombre est impair, il s'agit d'une version non testée, créée un jour donné. Les versions à troisième nombre impair ne veulent rien dire sans leur jour de création. Les versions à troisième nombre impair permettent de montrer un aperçu de la prochaine version paire. Ces versions ne sont jamais recommandées. Avec on a la possibilité de créer des Composants de LAZARUS avec une grammaire FREE PASCAL récente.

Le quatrième chiffre ou nombre montre une création d'une version stable de LAZARUS, à partir d'une base plus récente. En fait le quatrième chiffre ne vous apporte que la réparation de Bogues ou erreurs trouvées, ce qui est déjà intéressant, rien de nouveau sinon. Si le quatrième chiffre est impair la version de LAZARUS n'a pas été testée. Elle a cependant de grandes chances d'être suffisamment stable.

Il faut attendre ou participer à une nouvelle version de LAZARUS pour que le compilateur FREE PASCAL s'améliore, que les Composants standards LAZARUS s'améliorent.

Ne vous en faites pas pour l'attente, car de nouveaux Composants voient le jour chaque semaine ou chaque mois. En ajoutant ces Composants à LAZARUS vous

pouvez les étudier s'ils sont Open Source, afin de créer vos propres Composants si nécessaire.

L'évolution de votre Logiciel vers une nouvelle version de LAZARUS semble aisée. Il faudra cependant disposer des sources pour les Composants abandonnés, ou bien ceux ayant mal évolué, ce qui est cependant plus rare. C'est pour cela qu'il est intéressant de demander les sources des Composants, très souvent accessibles en PASCAL. Cela permet de rendre compatible ses Composants avec la dernière version de LAZARUS. Il n'y a souvent rien à faire.

Vous avez la possibilité de télécharger un "snapshot LAZARUS" WINDOWS ou REDHAT de la dernière version déboguée de LAZARUS. Pour les autres plates-formes, les Sources du "snapshot" permettent de créer un nouveau LAZARUS.

TÉLÉCHARGER LAZARUS

Téléchargez la dernière version de LAZARUS pour votre environnement, sur la page de téléchargement du projet LAZARUS à http://sourceforge.net/projects/lazarus.

Vous pouvez par la même occasion voir l'avancement du projet à http://www.lazarus-ide.org, dans la page "Roadmap".

Si vous souhaitez connaître l'état de la prochaine version allez à http://bugs.freepascal.org. Ce site en anglais permet aussi de savoir si une version d'au moins une semaine est utilisable dans vos Logiciels.

Vous pouvez aussi télécharger l'IDE CODE TYPHON, projet d'une entreprise privée. Ce clone, plus récent mais d'une entreprise cherchant des bénéfices monétaires, regroupe les Composants multi-plates-formes FREE PASCAL, à www.pilotlogic.com.

Il est possible de télécharger sa version LAZARUS du jour en tapant "snapshot lazarus" sur un moteur de recherche. Un snapshot sert à tester des Composants

LAZARUS en cours de publication.

Si vous souhaitez participer à LAZARUS il est possible de télécharger directement les Sources SVN sur le site Web de sourceforge à http://sourceforge.net/projects/lazarus, dans la partie "Develop".

INSTALLER LAZARUS

Sous LINUX

GNU LINUX centralise ses paquets de Logiciels. Ainsi un Logiciel GNU LINUX est très léger, car il utilise des librairies dans d'autres entités appelées Paquets. GNU signifie "GNU is Not UNIX". UNIX a été diffusé en semi-libre. La partie Libre d'UNIX c'est dorénavant GNU LINUX ou FREE BSD.

Sachez que LAZARUS utilise des librairies écrites en C, avec aussi de plus en plus d'assembleur. Vous devez disposer des librairies disponibles sur LINUX par défaut. LAZARUS nécessite le bureau GNOME, voire QT, aussi nommé KDE.

Pour les développeurs ou les débutants il est conseillé d'utiliser un GNU LINUX qui se met régulièrement à jour ou avec un système de paquet largement utilisé, comme LINUX MINT, EMMABUNTU, LINUX UBUNTU, LINUX DEBIAN. LINUX DEBIAN, avec son format de Paquets portant l'extension "deb", est la base de ces Distributions GNU LINUX.

Une version moins récente est disponible dans votre gestionnaire de paquets. Pour disposer de la version la plus récente possible, ajoutez le serveur de tests à votre gestionnaire de paquets. Avant, vérifiez si la version de www.sourceforge.net est plus récente que celle de votre gestionnaire. La communauté UBUNTU vous expliquera cette gestion de dépôts alternatifs appelés PPA.

Démonstration LAZARUS sous GNU LINUX

Afin de découvrir le gestionnaire de Paquets, une version de LAZARUS est dans le gestionnaire de paquets. Il suffit d'installer le paquet "LAZARUS" graphiquement. Ou bien tapez cette commande sur DEBIAN en mode "root" (Administrateur) :

```
apt install lazarus
```

Sur DEBIAN ou d'autres LINUX, pour accéder au mode "root", tapez su puis Entrée. Puis tapez les commandes "root". C'est en quittant par la Commande "exit" ou en fermant le Terminal que l'on sécurise son ordinateur des intrus.

Sur UBUNTU ou d'autres LINUX, le mode "root" s'obtient en précédant une commande par le mot "sudo", puis un espace évidemment. Une horloge que le programmeur appelle Timer va alors déterminer la sortie automatique de l'Administration par la commande.

Installation de la dernière version stable

En fonction de la Roadmap du site web bugs.freepascal.org, téléchargez la dernière version de LAZARUS sur lazarus-ide.org, puis sur www.sourceforge.net.

Après avoir désinstallé tous les paquets fp- et lazarus, on installe le ou les quelques paquets par l'outil visuel ou par cette commande DEBIAN en mode "root" :

```
apt remove fp-*
dpkg -i liens-vers-vos-paquets-fpc et-lazarus
```

Avec le sytème de Paquet, on télécharge ensuite les éventuelles librairies nécessaires :

```
apt install -f
```

Sous WINDOWS

LAZARUS s'installe en vous demandant de relier les fichiers PASCAL ou LAZARUS au Logiciel LAZARUS. Notez le répertoire d'installation de LAZARUS.

Mettre à jour sous WINDOWS

Avant de mettre à jour LAZARUS vers une nouvelle version, supprimez je répertoire "C:\lazarus" sous WINDOWS.

Sous MAC OSX

Le MAC a été créé en PASCAL. Puis l'environnement a été réécrit en C pour donner MAC OS.

LAZARUS utilise les interfaces MAC nommées CARBON et les nouvelles interfaces COCOA. Les nouvelles versions de MAC OSX n'utilisent que COCOA.

Pour installer LAZARUS vous devez avant télécharger XCode, nommé aussi "Developer Tools". Votre XCode doit correspondre à la version de votre MAC OSX.

Si vous disposez d'un vieux MAC OSX, vous devez chercher quelle version de MAC OSX vous avez. Au démarrage, allez en haut à gauche dans la pomme puis dans le menu À propos.

Cherchez alors sur un moteur de recherche "Xcode MAC OSX votre.version.de.MAC" afin de trouver la version de Xcode nécessaire. Puis cherchez sur un moteur de recherche comment télécharger XCode : "XCode votre.version.de.xcode".

Une fois que XCode est installé il suffit d'ouvrir les fichiers dmg téléchargés sur le lien sourceforge de lazarus-ide. Ensuite installez le paquet "pkg", d'abord de Free PASCAL Compiler ou fpc, puis de LAZARUS.

Créez vos scripts d'installation UNIX, afin de gagner du temps. MAC OSX est un UNIX-BSD bridé.

Configurer LAZARUS

LAZARUS peut se recompiler presque entièrement par défaut. Allez dans le menu "Outils" puis "Configurer la création de LAZARUS". Choisissez le profil "Nettoyer + Tout créer". Il faudra retrouver le profil "EDI Normal" ensuite.

"Nettoyer et tout créer" est à choisir lorsque des Sources LAZARUS en version d'essai ont été installées. Le nettoyage consiste à supprimer les unités compilées de la plate-forme. Pensez à modifier le répertoire de LAZARUS dans "Configuration" puis "Options".

LAZARUS utilise aussi les paquets avec un gestionnaire de paquets en ligne nommé onlinepackagemanager. Ainsi vous reconstruisez l'exécutable de LAZARUS en permettant d'installer de nouveaux Composants, c'est à dire ce qui composera votre Logiciel.

"Construire l'EDI avec les paquets"

Sauvegarder LAZARUS

Tout informaticien digne de son métier doit sauvegarder ses créations et ses installations en les dupliquant sur un autre support.

Pour sauvegarder LAZARUS il suffit de copier-coller son dossier de personnalisation.

Le répertoire de personnalisation stocke notamment les paquets installés dans des fichiers XML commençant par miscellaneous et packages. Il suffit alors de sauvegarder ses composants téléchargés stockés dans un dossier spécifique pour restaurer rapidement LAZARUS.

Sur GNU LINUX

Allez dans votre "Dossier personnel" ou bien sur un raccourci de dossier contenant votre nom ou prénom.

Toutes les informations personnelles de LINUX sont dans le dossier racine "/home". Il est d'ailleurs intéressant qu'il soit seul sur une très grosse partition. La partition obligatoire principale "/" ne nécessitera que 17 à 40 Go de place sans "/home", place suffisante pour installer des applications.

Il est préférable de créer une grande partition "/home" pour les données personnelles. Cela permet d'installer n'importe quel LINUX. Une partition LINUX obligatoire est swap et nécessitera autant de place que la mémoire vive sur votre ordinateur. C'est une mémoire de travail.

Tous les fichiers et dossiers cachés sont identifiés par un point en premier caractère. Dans votre navigateur de fichier LINUX accessible dans les raccourcis, c'est en général le menu "Affichage" qui permet de montrer les fichiers et dossiers cachés.

Le dossier caché "/home/mon-compte/.lazarus" est organisé pareillement à celui de WINDOWS. Seuls vont changer l'exécutable LAZARUS compilé et les chemins de binaires.

Le séparateur de chemin est inversé sur LINUX, comme vous le voyez ci-avant. Vous n'avez pas non plus de lettre de lecteur, plus ennuyante que réellement utile. Par contre les premiers répertoires commençant par "/" sont les répertoires installés par le système. Il existe une constante "DirectorySeparator" permettant le passage de WINDOWS à UNIX pour créer vos chemins LAZARUS.

Sur WINDOWS

Allez dans votre dossier de compte personnel. Dans le menu WINDOWS, c'est le menu avec votre prénom ou nom.

Ensuite il s'agit de montrer les dossiers et fichiers cachés. C'est soit dans Organiser ou Personnaliser. Comme WINDOWS évolue peu, c'est l'organisation de WINDOWS qui fait croire qu'il change.

Dans les données d'applications vous pouvez alors chercher "lazarus". Le répertoire caché "lazarus" est à copier-coller sur un stockage externe, comme une clé USB par exemple.

TÉLÉCHARGER LA TOUTE DERNIÈRE VERSION

Les développeurs LAZARUS fournissent des Release Candidate pour de bonnes raisons. Ils veulent savoir si leur version stable va être opérante sur l'ensemble des architectures.

Il peut arriver que des bugs passent les mailles des filets des nouvelles versions majeures au quatrième chiffre peu élevé. Des versions plus stables sont créées avec les versions mineures.

CRÉER UN LOGICIEL

Après avoir créé dans "Fichier" un "Nouveau" projet "Application" allez dans l'"Éditeur de Source".

Nouvelle "Application" LAZARUS

Vous voyez une Source créée automatiquement, ainsi qu'une fenêtre qui ne réagit pas comme une fenêtre habituelle. C'est votre base de travail permettant de créer un programme fenêtré, avec une grille en pointillés permettant de travailler graphiquement.

Allez dans le menu "Projet" puis dans "Inspecteur de Projet".
Vous voyez les fichiers qui ont été créés automatiquement. Le fichier commençant par "Unit1" c'est le formulaire de votre nouveau projet en Source PASCAL Objet.

Vous voyez aussi un fichier ".lpr". Cette extension de fichier signifie LAZARUS Project Resources, ou Ressources d'un Projet LAZARUS. Ce fichier, liant le projet vers les ressources LAZARUS, permet de compiler et d'exécuter votre formulaire.

Pour compiler et exécuter ce projet vide, cliquez sur le triangle flèche en bas à gauche de la barre d'outils LAZARUS. Vous voyez la même fenêtre vide sans les pointillés de présentation. C'est votre formulaire qui est exécuté. Vous pouvez le déplacer, l'agrandir, le diminuer et le fermer, contrairement au formulaire précédent.

Il s'agit bien d'un programme exécuté. Pour exécuter ce programme avec ce formulaire le Compilateur FREE PASCAL a compilé le formulaire grâce à ces Sources et celles de LAZARUS.

LAZARUS c'est un Environnement de Développement Intégré. Vous avez eu peu de programmation à effectuer pour créer ce Logiciel.

Le nom d'unité

Allez dans l'"Éditeur de Sources". Placez-vous au début de l'unité du formulaire.

Du Code PASCAL Objet commence toujours par la clause "Unit" reprenant lettre pour lettre le nom du fichier PASCAL. Le nom d'unité se change dans "Fichier", puis "Enregistrer sous".

En général chaque instruction PASCAL est terminée par un ";". Il existe de rares exceptions pour lesquelles le Compilateur vous avertit.

Pour la compatibilité UNIX ou LINUX il est préférable de renommer les noms d'unités en minuscules.

La clause "uses"

Ensuite vous avez sûrement à réutiliser des existants. La clause "uses" est faite pour cela. Elle permet de chercher les existants créés par l'équipe LAZARUS ou bien par un développeur indépendant de LAZARUS.

Lorsque vous ajoutez un Composant grâce à l'onglet des Composants cela ajoute aussi l'unité du Composant dans la clause "uses". Aussi un lien est créé dans le projet vers le regroupement d'unités du Composant. Ce regroupement d'unités est nommé "paquet" ou "package" en anglais. Ce paquet devient une condition du projet.

L'exemple

L'exemple consiste à créer une fenêtre permettant d'afficher "Hello World !".

Nous allons éditer un nouveau projet.

Démarrez LAZARUS.
Allez dans "Fichier" puis "Nouveau". Choisissez "Application".

Tout d'abord nous allons rechercher l'unité "Dialogs" avec beaucoup de fainéantise.
Placez-vous après une virgule de la clause "uses". Comme chaque instruction, Cette dernière se termine par un ";".
En respectant la présentation LAZARUS, tapez uniquement "di".
Ce qui va suivre ne fonctionne que si votre Code Source est correct, du début de l'unité jusqu'à votre ligne éditée.
Appuyez en même temps sur "Ctrl" avec la barre d'espacement. Relâchez. Il s'affiche une liste d'unités LAZARUS. Choisissez "dialogs" avec les flèches et la touche "Entrée". N'oubliez pas de placer ensuite une virgule pour ne pas créer d'erreur de compilation. Vous avez ajouté l'unité Dialogs.

Pour voir ce que vous avez ajouté, maintenez enfoncé "Ctrl" tout en cliquant sur l'unité "Dialogs". L'unité "Dialogs" s'affiche. Vous allez vers la Source liée en faisant de cette manière. LAZARUS va chercher ce qu'il connaît grâce aux liens de vos projets et un Code correct, avec de la mémoire aussi.

On voit que la clause "uses" de "Dialogs" contient d'autres unités. Celles-ci sont aussi ajoutées à votre projet, si elles ne l'étaient pas déjà. En effet des unités obligatoires sont toujours utilisées.

Toute écriture de Code ajoute des informations à votre exécutable, avec en plus des unités qui sont intégrées. Votre programme compilé contient beaucoup de Code, mais il est dépendant de peu de librairies. En effet, votre exécutable aura intégré les sources incluses que vous pouvez consulter.

Grâce aux onglets de l'"Éditeur de Sources", vous pouvez retourner sur votre Source en cliquant sur l'onglet "Unit1".

Ce nom "Unit1" n'est pas explicite. Nous allons le renommer en gardant une partie des informations importantes.

Cliquez dans le menu LAZARUS sur "Fichier" puis "Enregistrer". Créez votre répertoire de projet, puis sauvegardez votre unité en commençant par "u_". Cela indique comme avant que l'on sauvegarde une unité. Les fichiers sont ainsi regroupés au même endroit. Ensuite on met le thème de l'unité : "hello". Cela donne "u_hello". Sauvegardez.

Si vous avez mis des majuscules dans le nom d'unité, LAZARUS peut vous demander de renommer ce nom d'unité en minuscule. N'allez que pas contre ces recommandations.

Votre nom d'unité est donc "u_hello". Elle peut être rajoutée dans une clause "uses" d'une autre unité du projet.

La présentation

Nous allons rajouter un bouton à notre formulaire. Pour aller sur le formulaire de l'unité "u_hello" appuyez sur "F12", quand vous êtes sur l'unité dans l'"Éditeur de Source". Cliquez de nouveau sur "F12" permet de revenir sur l'"Éditeur de Source". Revenez sur la fiche.

Allez sur la palette des Composants. Sélectionnez dans le premier onglet "Standard" le bouton "TButton". Cliquez sur l'icône "TButton". Il s'enfonce.

Cliquez sur le formulaire pour placer votre "TButton". Vous voyez à l'exécution la même fenêtre avec son "TButton" sans les points noirs. Ces points noirs servent à se repérer à la conception.

Cliquez sur le bouton de votre formulaire puis sur "F11". Vous sélectionnez l'"Inspecteur d'Objets", qui permet de modifier votre "TButton".

Vous êtes sur l'onglet "Propriétés" de l'"Inspecteur d'Objets". Cet onglet classe les propriétés par ordre alphabétique.

Vous pouvez changer la propriété "Name" de votre "TButton", afin de renommer votre "TButton" en "BCliquez". Vous voyez alors le texte du "TButton" changer. C'est une automatisation de ce Composant, qui change sa propriété "Caption" quand on change pour la première fois son nom.

Pour vous rendre compte, allez sur la propriété "Caption" et enlevez le "B". Le nom de votre Composant n'a pas changé. Pourtant la propriété "Caption" a bien changé le texte de votre "TButton".

Le Code Source

Allez sur l'onglet "Événements" de l'"Inspecteur d'Objets" par F11. Double cliquez sur l'événement "OnClick". Cela fait le même effet que d'appuyer sur les trois points : Vous créez un lien vers une procédure dans votre "Éditeur de Source".

Tapez entre le "Begin" et le "End;" de cette nouvelle procédure deux espaces puis "show". Puis appuyez en même temps sur "Ctrl" suivi de "Espace".

Si vous avez ajouté l'unité "Dialogs" vous voyez la procédure "ShowMessage". Cette procédure simple possède un seul paramètre chaîne. Sélectionnez cette procédure avec les flèches et entrée. Vous pouvez aussi utiliser la souris.

Vous avez maintenant à ajouter le paramètre permettant d'afficher du texte. Les paramètres des procédures sont intégrés grâce aux parenthèses.

Ouvrez votre parenthèse. Les chaînes sont ajoutées en les entourant par un caractère "'". L'instruction se termine par un ";". Écrivez donc :

```
Begin
 ShowMessage ( 'Hello World' );
End;
```

Nous allons afficher un message "Hello World" quand on clique sur le bouton "BCliquez".

Appuyez simultanément sur "Ctrl" suivi de "F9". Cela compile votre exécutable, en vérifiant la syntaxe et en donnant des avertissements.
Cliquez sur ou appuyez sur "F9" pour exécuter votre Logiciel fenêtre.
Après avoir appuyé sur le bouton voici ci-après ce que cela donne :

Un exemple avec un événement

La Source du formulaire

Dans la source de votre formulaire, le type égal à "class (TForm)" décrit l'élément essentiel de l'EDI LAZARUS. Il s'agit de la fenêtre affichée contenant le bouton. On ne voit pas dans cette déclaration l'endroit où on a placé le bouton. L'endroit où on a placé le bouton se modifie dans l'"Inspecteur d'Objets".
L'"Inspecteur d'Objets" sauve les propriétés dans le fichier portant l'extension ".lfm". Vous pouvez voir ce fichier en affichant sur le formulaire modifiable un menu, grâce au bouton droit de la souris. Dans ce menu cliquez sur "Afficher le code Source".

"Afficher le Source" dans la conception du formulaire

En scrutant le fichier ".lfm" vous voyez le bouton et ses coordonnées affichées.

Le type égal à "class (TForm)" permet d'afficher la fenêtre et son bouton en lisant ce fichier ".lfm".

Les modifications apportées au fichier permettent entre autres d'intervertir des Composants. Pour faire cela vous avez relié le paquet du Composant au projet en l'ajoutant comme condition au "Gestionnaire de projet".

PAQUETS SPÉCIFIQUES

Dans la version 0.9.30 de LAZARUS a été ajouté un paquet, une bibliothèque donc, permettant de faciliter le développement pour les débutants.

Ce paquet installe des fonctionnalités LAZARUS permettant de s'habituer au nommage, tout en facilitant la création de Code au sein de LAZARUS.

Allez dans "Paquets", puis "Configurer les paquets". Une fenêtre s'ouvre.
Sur la liste de droite, sélectionnez le paquet "educationlaz", pour l'"Installer". "Reconstruisez LAZARUS".

Si le paquet n'était pas compilé, il s'agit de le recompiler dans le menu "Paquets" puis "Ouvrir le paquet chargé". Si un paquet ne recompile toujours pas, effacez le répertoire "lib" de la personnalisation de LAZARUS (cf **Sauvegarder LAZARUS** dans le chapitre sur LAZARUS).

Après que LAZARUS ait redémarré, dans "Outils", puis "Options", vous pouvez changer les paramètres pour débutants dans "Éducation".

Dans les options d'"Education", dans "Général", activez le mode "Débutant".

Dans ces options toujours et dans la "Palette de Composants" "Montrez tout". Il est en effet possible d'y cacher les Composants que l'on ne souhaite pas utiliser. Mais cette option, activée pour les professeurs, est surtout réservée aux démonstrations.

Vous pouvez afficher ou cacher d'autres outils, comme les boutons de référencement d'unités, de classes.

Dans la version 1.8 de LAZARUS, a été installé le paquet onlinepackagemanager, qui permet d'installer facilement des paquets. Il sera probablement intégré à LAZARUS plus tard.

INDENTATION PASCAL

Vous voyez dans l'exemple que le Code Source est joliment présenté. Cela n'est pas une futilité puisque vous avez eu envie de le regarder. Vous avez peut-être eu envie de prendre exemple sur ce qui avait été fait. Voici la recette à suivre pour faire de même et mieux encore.

Un Code Source bien présenté ce sont des développeurs avec un comportement clair et précis. L'indentation c'est la présentation du Code Source pour qu'il devienne lisible. En effet vous créez du Code machine, alors que ce sont des hommes et femmes qui le scrutent, et le modifient correctement. Les erreurs sont toujours humaines.

L'indentation c'est donc du Code Source lisible. L'indentation permet de présenter le Code Source afin de le comprendre comme clair, sans avoir à fouiller pour trouver ce que l'on cherche.

L'indentation des éditeurs PASCAL est simple. En voici un résumé :
Chaque instruction ou nœud (var, type, class, if, while, case, etc.) est à la même position que la ligne précédente.
Chaque imbrication d'un nœud doit être décalée de deux espaces vers la droite.
Les instructions incluses entre les "Begin" et "End ;" sont décalées de deux espaces vers la droite.

Cette présentation permet de revoir du Code Source plus facilement. Les développeurs qui regardent du Code Source avec cette présentation sont plus précis et plus sûrs d'eux.

Pour aider à indenter deux combinaisons de touches sont très importantes dans les éditeurs PASCAL :
Ctrl + K + U permet de décaler votre sélection de lignes de deux espaces vers la gauche.
Ctrl + K + I permet de décaler votre sélection de lignes de deux espaces vers la droite.

Vous pouvez aussi ajouter des règles de présentation du Code Source qui facilitent la maintenance. Nous vous disons pourquoi appliquer ces règles de présentation :
Des commentaires sont obligatoirement avant chaque Code Source de fonctions ou procédures. Ainsi les commentaires avant chaque fonction ou procédure sont vus dans l'éditeur FREE PASCAL lorsque vous mettez la souris sur la fonction ou procédure.
Des commentaires sont dans le Code Source si on y trouve un Bogue ou erreur. Si on corrige un Bogue dans le Code Source, il est possible que l'erreur revienne s'il n'y a pas de commentaires.
Des commentaires sont dans le Code Source dès que l'on réfléchit pour écrire une Source. Si le Code demande réflexion pour être fait, c'est que vous y réfléchirez à nouveau si vous n'y mettez pas de commentaires. Notre mémoire oublie ce que l'on a écrit.
Chaque fonction doit tenir sur la hauteur d'un écran. Une fonction qui ne tient pas sur la hauteur d'un écran est non structurée, illisible, difficile à maintenir, difficilement compilable si retravaillée.

Pour créer des commentaires sélectionnez le texte à commenter, puis appuyez sur Shift+Ctrl+V. Shift+Ctrl+U permet de dé-commenter.

Il existe d'autres méthodes de structuration, comme aligner des séries d'instructions mot à mot. Cela permet de retrouver facilement les différences si le Code Source se répète.

Comme l'éditeur FREE PASCAL peut terminer les mots, il est possible d'affecter des noms longs de variables et procédures.

Toute nouveauté de l'éditeur est à scruter. Cela facilite le travail. Vous pouvez définir votre présentation et la tester.

Les exemples que nous vous montrons après sont indentés.

Paramétrer l'indentation

Vous pouvez modifier les "Options" d'"Indentation" dans "Outils", puis "Options", puis "Éditeur", puis "Général", puis "Indentation".

STRUCTURE DU CODE SOURCE

Nous allons décrire le Code Source précédemment créé.

En général une unité faite en PASCAL se structure avec d'abord un en-tête, puis des déclarations publiques, puis le Code Source exécutable. Toute unité PASCAL se termine par "end.".

La déclaration Unit

La déclaration "Unit" est la première déclaration d'un fichier PASCAL. Elle est obligatoire et obligatoirement suivie du nom de fichier PASCAL, sans l'extension. On change cette déclaration en enregistrant l'unité.

Les déclarations publiques

Les déclarations publiques comprennent obligatoirement le mot clé "interface" indiquant le début des déclarations publiques.

Vous pouvez ensuite y placer l'instruction "uses" permettant d'utiliser dans les déclarations publiques les unités déjà existantes. À tout moment vous pouvez ajouter une unité dans cette clause grâce au menu "Source".

Vous pouvez ensuite placer toute déclaration constante, type, variable, fonction

ou procédure. Toute déclaration incluse dans la compilation ajoute du Code à l'exécutable.

Les variables allouent soit un pointeur soit un type simple. Les types simples sont faciles à utiliser.
Un pointeur est une adresse pointant vers un endroit de la mémoire. Il permet de définir et de stocker des types complexes.
Les constantes publiques peuvent être centralisées dans une ou plusieurs unités.

Le Code Source exécuté

Le Code Source commence par le mot clé "implementation".
En plus des déclarations déjà définies, on place dans cette partie le Code Source exécuté grâce aux fonctions et procédures.

Dans le Code Source exécuté on peut placer les mêmes déclarations que celles données précédemment. Seulement elles ne sont plus publiques donc pas utilisables ailleurs.

La procédure "ShowMessage" vous a permis d'afficher une boîte avec un message et un bouton.
LAZARUS est partagé. Vous pouvez donc scruter ses sources. Si vous regardez le Code Source de "ShowMessage" en cliquant plusieurs fois avec "Ctrl", vous voyez qu'elle est déclarée avec le mot clé "procedure".

Une fonction s'appelle de la même façon mais renvoie une variable retour. En effet le rôle des fonctions et procédures est de modifier une partie de votre Logiciel. Elles transforment une entrée en sortie comme le ferait une équation mathématique.

L'événement que vous avez créé est une procédure particulière qui a permis d'afficher votre message à l'écran. La sortie ici c'est l'affichage de "Hello World" dans une fenêtre de dialogue.

Toute fonction ou procédure déclarée publiquement doit avoir une correspondance dans le Code Source exécuté. Le Compilateur vous le rappelle si vous oubliez. Le Code Source que vous avez ajouté était dans la partie "implementation".

Fin de l'unité

La fin de l'unité est définie par "end."

LES FICHIERS RESSOURCES

À la fin d'un fichier PASCAL contenant votre formulaire, voici ce que l'on peut mettre :

```
{$IFDEF FPC}
 {$i Resources.lrs}
{$ENDIF}
```

La combinaison Shift+Ctrl+D permet d'entourer toute source avec un IFDEF à définir.

Ces trois lignes signifient : Si la directive FPC ou Compilateur FREE PASCAL est activée, alors inclure le fichier "Resources.lrs". Le chapitre **De PASCAL vers FREE PASCAL** décrit les **IFDEF** en détail.

LAZARUS utilise d'autres fichiers ressources que Turbo PASCAL ou DELPHI. On appelle les fichiers ressources LAZARUS les fichiers ".lrs". Ils nécessitent idéalement d'avoir des fichiers images en format image compressée XPM. Les fichiers ".lrs" peuvent contenir aussi le fichier ".lfm" dans les vieilles unités de formulaires, pour les versions de LAZARUS inférieures à 0.9.29.

Nous vous expliquons plus loin la création de votre premier Composant. Une partie de ce chapitre décrit la création et l'utilisation du fichier ".lrs".

RETRAVAILLER LES SOURCES

LAZARUS aide à modifier les Sources de vos Logiciels. N'oubliez pas dans l'"Éditeur de source", quand vous cliquez à droite sur un mot de vos sources, le sous-menu "Retravailler". Vous pouvez par exemple changer l'identificateur de tout nom de fonction ou variable. Cela permet d'avoir des sources plus lisibles.

TOUCHES DE RACCOURCIS POUR COMPLÉTION

La complétion permet d'écrire à votre place. C'est indispensable pour tout programmeur cherchant à gagner du temps.

Ctrl+Espace	Rechercher dans le Code lié :
Ctrl+Maj +Espace	Afficher la syntaxe des méthodes :
Ctrl+Maj+ flèche haut ou bas	Dans l'éditeur : Aller directement de la déclaration d'une méthode à son implémentation.

Ctrl+Maj+C	Complément de Code : vous entrez **procedure** toto(s:**string**) dans :
	`type`
	` TForm1 = class(TForm)`
	` private`
	` procedure toto(s:string);`
	` public`
	Il vous écrit, dans la partie implémentation :
	`procedure TForm1.toto(s:string);`
	`Begin`
	
	`end;`

Ctrl+Maj+i ou Ctrl+Maj+u	Indenter plusieurs lignes à la fois Passer de **Begin** **begin** **if** X >0 **then** x=0; **if** X >0 **then** x=0;

```
if X<0 then x=-1;              if X<0 then x=-1;
A:=truc+machintruc;            A:=truc+machintruc;
 end;                     end;
```

	Sans le faire ligne par ligne ? Sélectionner les lignes puis faire Ctrl+Maj+i pour déplacer les lignes vers la droite, ou Ctrl+Maj+u pour les déplacer vers la gauche.
Maj+touche flèchée et Ctrl+touche flèchée	Positionner ou dimensionner au pixel près un Composant

Ctrl+j	Faire appel aux modèles de Code.
	Nota : les modèles de Code permettent d'écrire du Code tout fait par exemple: en choisissant le modèle de Code if then else après avoir fait Ctrl + j, on obtient
	`if then`
	`begin`
	
	`end`
	`else`
	`begin`
	
	`end;`
	
	Autre astuce : chaque modèle de Code possède une abréviation. Tapez cette abréviation puis Ctrl + j. Tapez par exemple "if" puis Ctrl + j. Les lignes correspondant à "if then else" s'écrivent à votre place. Les abréviations disponibles sont visibles en faisant Ctrl + j.

Ctrl+Maj + 1 à 9 et Ctrl+ 1à9	Placer des marques (des signets) dans un Source pour pouvoir y revenir ultérieurement Vous êtes sur un bout de Source et vous vous aller voir ailleurs dans l'unité puis revenir rapidement : Tapez :CTRL SHIFT 1 (ou un chiffre de 1 a 9 au-dessus des lettres et non dans le pavé numérique) l'éditeur met un "1" dans la marge. Pour revenir vous faites CTRL avec 1 Pour annuler la marque, soit vous vous placez sur la

	ligne et vous refaites CTRL SHIFT 1, soit vous vous placez ailleurs et vous refaites CTRL SHIFT 1, pour déplacer la marque.
Ctrl+ flèche droite ou gauche	Se déplacer au mot suivant ou précédent.
Ctrl+Maj + droit ou gauche	Se déplacer au mot suivant ou précédent tout en sélectionnant.
Ctrl +haut ou bas	Revient au même que d'utiliser l'ascenseur

TOUCHES DE RACCOURCIS POUR LE PROJET

Touche	Action du menu
Ctrl+F11	Fichier | Ouvrir un projet
Maj+F11	Projet | Ajouter au projet
F9	Exécuter | Exécuter
Ctrl + F9	Compiler le projet
Ctrl+S	Fichier | Enregistrer
Ctrl+F2	Réinitialiser le programme = arrête votre programme et revient en mode édition
Ctrl+F4	Ferme le fichier en cours

TOUCHES DE RACCOURCIS DE VISIBILITÉ

Touche	Action du menu
Ctrl+F3	Voir | Fenêtres de débogage | Pile d'appels
F1	Affiche l'aide contextuelle
F11	Voir | Inspecteur d'Objets
F12	Voir | Basculer sur Formulaire/Unité
Ctrl+F12	Voir | Unités
Maj+F12	Voir | formulaires

Ctrl+Maj+E	Voir l'explorateur de Code
Ctrl+Maj+B	Voir l'explorateur de classe
Ctrl+ Maj+T	Ajouter un élément à faire
Alt+F10	Affiche un menu contextuel
Alt+F11	Fichier | Utiliser l'unité
Alt+F12	Bascule Form visuel / Source de la forme

TOUCHES DE RACCOURCIS DE DÉBOGAGE

Les raccourcis de débogage sont généralement utilisés dans l'éditeur de Source. Ils permettent de scruter le Code Source. Il est possible de placer un point rouge sur une ligne de votre Source, dans la marge grisée à gauche de votre Source. C'est un point d'arrêt. Il va être appelé dès que l'on exécutera la ligne avec ce point d'arrêt.

Exercice

Placez un point d'arrêt à "ShowMessage" dans votre exemple.
Exécutez et cliquez sur le bouton. En général le programme se fige, puis LAZARUS se place sur le "Point d'arrêt" que vous avez mis.

Mes points d'arrêt sont inactifs

Pour que ces points d'arrêt fonctionnent, votre projet doit se compiler d'une certaine manière. Votre exécutable devient ainsi dix fois plus important qu'habituellement. Si les points d'arrêt fonctionnent, votre exécutable est très lourd.

Allez dans le menu "Projet", puis "Options du Projet", puis "Options de compilation".

Allez dans l'onglet "Débogage". Cochez en haut "Générer les informations de débogage" et "Afficher les numéros de lignes". Les infos de débogage et les numéros de ligne servent à ce que le débogueur se repère.

Allez dans l'onglet "Compilation et édition des liens". En bas dans "Optimisations" choisissez l'option "Niveau 0" ou "Niveau 1". Les niveaux 0 et 1 créent un très gros exécutable. Le niveau 1 le fait en mode débogage. Mais on scrute alors les points d'arrêt.

ALLÉGER L'EXÉCUTABLE

Ouvrez le menu "Projet", puis "Options du Projet", puis "Options de compilation".
Ouvrez l'onglet "Débogage". Décochez en haut "Générer les informations de débogage". Ces infos et numéros de lignes supplémentaires alourdissent votre exécutable.
Ouvrez l'onglet "Compilation et édition des liens". En haut dans "Optimisations" choisissez le dernier niveau d'optimisation.

Le niveau 1 par défaut crée un gros exécutable

TOUCHES DE RACCOURCIS DU DÉBOGUEUR

Touche	Action du menu
F4	Exécuter | Jusqu'au curseur
F5	Exécuter | Ajouter un point d'arrêt
F7	Exécuter | Pas à pas approfondi
Maj+F7	Exécuter | Jusqu'à la prochaine ligne
F8	Exécuter | Pas à pas
Ctrl+F5	Ajouter un point de suivi sous le curseur
Ctrl+F7	Évaluer/Modifier

TOUCHES DE RACCOURCIS DE L'ÉDITEUR

Touche	Action
Ctrl+K+B	Marque le début d'un bloc
Ctrl+K+C	Copie le bloc sélectionné
Ctrl+K+H	Affiche/cache le bloc sélectionné
Ctrl+K+K	Marque la fin d'un bloc
Ctrl+K+L	Marque la ligne en cours comme bloc
Ctrl+K+N	Fait passer un bloc en majuscules
Ctrl+K+O	Fait passer un bloc en minuscules
Ctrl+K+P	Imprime le bloc sélectionné
Ctrl+K+R	Lit un bloc de **procedure** puis un fichier
Ctrl+K+T	Marque un mot comme bloc
Ctrl+K+V	Déplace le bloc sélectionné
Ctrl+K+W	Écrit le bloc sélectionné dans un fichier
Ctrl+K+Y	Supprime le bloc sélectionné
Shift+Ctrl+V	Le bloc devient un commentaire
Shift+Ctrl+U	Le bloc redevient une source
Shift+Ctrl+N	Entourer la sélection
Shift+Ctrl+D	Entourer par un **IFDEF**
Shift+Ctrl+C	Compléter la source
Shift+Ctrl+G	Insérer un GUID (Voir le chapitre sur les **Composants**)

TOUCHES DE RACCOURCIS DE L'ENVIRONNEMENT

Touche	Action du menu
Ctrl+C	Édition | Copier
Ctrl+V	Édition | Coller

Ctrl+X   Édition | Couper

MA PREMIÈRE APPLICATION

INTRODUCTION

Avant de créer sa première Application, sachez avant si c'est utile de créer le Logiciel voulu. Vérifiez que ce que vous voulez ne se fait pas déjà. A l'heure actuelle, seuls les Logiciels avec une personnalisation accrue sont à créer.

Dans notre premier exemple, nous allons créer un Logiciel de conversion de fichiers PASCAL qui permet de facilement changer ses Composants dans tout son projet.

Un Logiciel doit être créé rapidement. Nous allons rapidement créer une interface Homme-Machine. LAZARUS est un outil permettant de créer rapidement un Logiciel, grâce à son interface graphique et du Code uniquement dédié au développement.

LES COMPOSANTS

Pour créer rapidement son Logiciel, il suffit de placer correctement les Composants, après avoir cliqué sur un Composant de la palette de Composants. La "Palette de Composants" comporte un ensemble d'onglets identiques à des onglets de classeur. Lorsque vous cliquez sur un onglet portant un certain nom, un ensemble de Composants classés sous ce nom s'affichent.

La palette de Composants

Vous voyez qu'il est possible de placer un certain nombre de Composants, sans limite de présentation.

Certains Composants ne sont représentés que par le même graphisme de même taille que son icône situé dans la palette. Par exemple vous avez dans l'onglet "Standard" le premier Composant, qui est un Composant de menu. Ces Composants possédant un visuel non changeant n'ont en général aucune propriété graphique immédiate. Ce sont des Composants invisibles capables de réaliser certaines opérations visuelles ou pas.

D'autres Composants ajoutent un visuel à votre formulaire. Ce sont des Composants visuels. Avant de placer vos Composants visuels, n'oubliez pas d'utiliser un Composant "TPanel" pour une présentation soignée. Les Composants "TPanel" permettent à vos Composants visuels de se disposer sur l'ensemble de l'espace visuel disponible.

CHERCHER DES PROJETS

Ceci est une aide servant à améliorer vos projets grâce à INTERNET.

LAZARUS possède à son installation un nombre limité de Composants. Il est possible de trouver des savoir-faire sur INTERNET. Pour trouver un Composant, définissez ce que vous souhaitez, ainsi que les alternatives possibles. Ce n'est qu'avec l'expérience que l'on trouve des Composants adéquates.

Sur votre moteur de recherche tapez plutôt des mots anglais comme "component", "project" ou "visual" avec "LAZARUS" voire "DELPHI". Puis, avec ces mots, tapez votre spécification en anglais. Changez de mots ou de domaines, si vous ne trouvez pas.

Le paquet intégré à LAZARUS onlinepakagemanager permet d'installer des composants LAZARUS très facilement. Une fois que vous savez quel composant installer, le composant Open Source peut être installé grâce au dernier menu des paquets.

Un site web de Composants LAZARUS s'appelle www.lazarus-components.org.

INSTALLER DES COMPOSANTS MANUELLEMENT

Si vous voulez installer une version spécifique de vos Composants, il s'agit de ne pas passer par le gestionnaire de paquets en ligne.

Pour installer des Composants manuellement, décompressez votre archive, puis placez-la dans un dossier qui ne bougera pas.
Avec LAZARUS ouvrez le ou les paquets portant l'extension ".lpk".
Compilez et installez. Ne recompilez LAZARUS que si vous n'avez plus de Composant à ajouter.

S'il manque des dépendances cherchez dans le dossier, dans le gestionnaire de paquets en ligne, onlinepackagemanager, ou sur Internet les paquets manquants. Installez les.

Si le paquet ne compile toujours pas en fonction d'une unité d'un autre paquet, recompilez le paquet de l'unité. Ou bien mettez à jour vos paquets, pour certains sur un outil de partage comme SVN. N'oubliez pas d'installer avant Tortoise ou Rapid SVN, voire Tortoise HG ou GIT. Si votre version de LAZARUS est trop récente, il est possible qu'il faille installer la version précédente de LAZARUS.

Si vous avez mis à jour votre paquet et que ça ne compile pas effacez le répertoire "lib" des paquets pour recompiler l'ensemble. Le répertoire lib est à effacer aussi quand on change d'interface graphique.

Pour tester vos Composants, avant de compiler LAZARUS, pensez à dupliquer l'exécutable LAZARUS.

L'EXEMPLE

Nous allons créer cette interface ci-après. Elle s'adapte automatiquement à sa fenêtre. Elle peut aussi être grossie par les caractères et images.

Une présentation soignée

Pour commencer notre Interface, dans "Fichier", créez une "Nouvelle application".
Puis disposez un Composant "Standard" "TPanel".
Les Composants de type "TPanel" ou de son type ascendant "TCustomPanel" permettent à l'interface d'utiliser le maximum d'espace de travail.

Notre premier "TPanel" sert à disposer les boutons. En cliquant sur l'onglet "Standard" vous pouvez ajouter un panneau, en anglais un "panel". Cliquez dessus, puis cliquez sur le formulaire.

Tapez sur F11 pour afficher l'inspecteur d'Objet. Votre "TPanel" y est sélectionné. Cliquez sur la boîte à option de la propriété "Align", et affectez "alBottom" à "Align".

Le deuxième "TPanel" sert à disposer le Composant de sélection du répertoire. Ajoutez le "TPanel" de la même manière mais ajoutez-le dans la zone vide.

Tapez sur F11 pour afficher l'"Inspecteur d'Objet". Votre "TPanel" y est sélectionné. Cliquez sur la boîte à option de la propriété "Align", puis affectez "alTop" à "Align".

Le troisième "TPanel" affiche les fichiers à convertir.
Disposez un deuxième "TPanel" dans la zone où il n'y a pas de "TPanel".

Votre "TPanel" doit donc s'aplatir sur tout le formulaire. Affectez "alClient" à la propiété "Align" de ce "TPanel" afin de disposer d'un maximum d'espace. Vous pouvez enlever les bordures "bvRaised" de vos "TPanel"...

L'Interface Homme Machine

Ensuite nous allons disposer les Composants de sélection de répertoire et de fichiers dans les "TPanel". Ils sont dans l'onglet "Misc". Nous allons aussi disposer les boutons.

Disposez un Composant "TDirectoryEdit" sur le panneau "TPanel" du haut. Le Composant "TDirectoryEdit" peut sembler insatisfaisant graphiquement. En effet, il n'affiche pas l'arborescence détaillée des répertoires.

Il est possible d'ancrer le Composant "TDirectoryEdit" sur la longueur du "TPanel". Ou alors choisissez un Composant de sélection de répertoire plus abouti. Les Composants fournis avec LAZARUS sont testés sur l'ensemble des plates-formes disponibles, en fonction de la "RoadMap" LAZARUS. Consultez-la sur www.lazarus-ide.org. Par exemple la "Roadmap" LAZARUS indique en 2012 que des Composants ne marchent pas sur MAC OSX version COCOA.

Disposez un Composant de sélection de fichiers "TFileListBox" dans le panneau du milieu. Vous pouvez lui affecter la valeur "alClient" à sa propriété "Align". Vos fichiers sont disposés sur l'ensemble de l'interface. Ils sont donc visibles pour l'utilisateur.

Disposez les boutons "TButton" sur le panneau du bas. Les boutons n'ont pas besoin d'être alignés, car le texte qui leur est affecté a une longueur connue. Ils sont alignés à partir de la gauche.

Vous pouvez par exemple créer un bouton qui affiche la date du fichier, son chemin, etc.

Le code

Nous allons créer un Logiciel de conversion de fichiers LAZARUS.

Créez un bouton avec le nom "Convertir", dans la propriété "name". Les boutons

sont dans l'onglet "Standard". Vous voyez que son libellé visuel change. Vous avez affecté aussi la propriété "Caption" du Composant, grâce au code réservé au développeur.

Il est nécessaire de relier l'éditeur de répertoire avec la liste de fichiers. Double-cliquez sur l'événement "OnChange" du "TDirectoryEdit". Placez-y cette Source :

```
procedure TForm1.DirectoryEditChange(Sender:
TObject);
begin
 FileListBox.Directory := DirectoryEdit.Directory ;
end;
```

Double-cliquez sur l'événement "Onclick" du bouton.
Placez-y cette Source :

```
procedure TForm1.ConvertirClick(Sender: TObject);
var li_i : Integer ;
begin
 For Li_i := 0 To FileListBox.Items.Count - 1 do
 Begin
   ConvertitFichier ( DirectoryEdit.Directory +
DirectorySeparator + FileListBox.Items.Strings [ li_i ],
FileListBox.Items.Strings [ li_i ] );
 End ;
end;
```

Ne compilez pas. Il manque la procédure "ConvertitFichier" ici :

```
procedure TForm1.ConvertitFichier(const FilePath,
FileName : String);
var li_i : Integer ;
 ls_Lignes : WideString ;
 lb_DFM ,
 lb_LFM : Boolean ;
 lst_Lignes : TStringList ;
begin
 lb_DFM := PosEx ( '.dfm', LowerCase ( FilePath ),
length ( FilePath ) - 5 ) >= length ( FilePath ) - 5 ;
 lb_LFM := PosEx ( '.lfm', LowerCase ( FilePath ),
```

```
length ( FilePath ) - 5 ) >= length ( FilePath ) - 5 ;
 lst_Lignes := TStringList.Create;
 try
 lst_Lignes.LoadFromFile ( FilePath );
 ls_Lignes := lst_Lignes.Text;

 Application.ProcessMessages;

  // Conversion d'une chaîne du fichier
    ls_Lignes    :=    StringReplace    (    ls_Lignes,
'TADOQuery',                              'TZQuery',
[rfReplaceAll,rfIgnoreCase] );

  if lb_DFM
 or lb_LFM Then
 Begin
 End;
 else
 Begin
 End;

 Application.ProcessMessages ;
 lst_Lignes.Text :=  ls_Lignes ;
 lst_Lignes.SaveToFile  ( FilePath );
 finally
 lst_Lignes.Free;
 end;
end;
```

Vous voyez que cette méthode prend beaucoup de place. Vous pouvez la couper, afin de mieux programmer.

Une fois que vous avez créé la procédure "ConvertitFichier", appuyez simultanément sur "Ctrl", "Shift", avec "C". Ainsi, si la source compile, la procédure "ConvertitFichier" est renseignée dans votre fiche en fonction de la déclaration.

Au début on teste si l'extension de fichier est "lfm", "dfm", ou "pas".

La fonction "ProcessMessages" de TApplication sert pendant les boucles. Elle

permet à l'environnement de travailler, afin d'afficher la fenêtre.

La fonction "StringReplace" retourne une chaîne, venant d'une sous-chaîne dont le paramètre a éventuellement été remplacée. Elle possède des options, comme le remplacement sur toute la chaîne, la distinction ou pas des majuscules et minuscules.

A la fin on affecte le "TStringlist" par sa propriété "Text", puis on remplace le fichier en utilisant la procédure "SaveToFile", avec le nom de fichier passé en paramètre de la procédure "ConvertitFichier". Faites une copie du projet, avant de le convertir par cette moulinette.

Il n'y a plus qu'à filtrer les fichiers de la liste de fichiers en fonction du résultat demandé par l'utilisateur : Convertir tout ou convertir des fichiers "lfm", des fichiers "dfm" ou des fichiers "pas".

Placez une TFilterComboBox. Renseignez les types de fichiers voulus en renseignant "Filter" grâce aux "...". Placez ce genre de filtre :

```
Tous les fichiers (*.*)|*.*|Fichiers DELPHI et
LAZARUS|*.dpr;*.pas;*.dfm;*.lfm|Fichiers
projet|*.dpr|Unités DELPHI|*.pas|Propriétés
DELPHI|*.dfm|Propriétés LAZARUS|*.lfm
```

On voit que l'éditeur de propriété permet de gagner du temps.

Puis renseignez l'événement "OnChange" avec cette Source :

```
procedure TForm1.FilterComboBoxChange(Sender:
TObject);
begin
 FileListBox.Mask := FilterComboBox.Mask;
end;
```

Vous pouvez maintenant vous amuser à changer les Composants de projets LAZARUS, en série.

Notre interface est prête à être renseignée. Ce qui est visible par l'utilisateur a été créé rapidement. Nous n'avons pas eu besoin de tester l'interface. Pour voir le

comportement de cet exemple, exécutez en appuyant sur "F9" ou en cliquant sur le bouton avec le triangle flèche.
Vous pouvez agrandir votre formulaire ou le diminuer pour tester votre interface.

CONCLUSION

Votre application peut maintenant être agrandie, y compris les fontes. La propriété ScaleDPI de LAZARUS 1.8 permet cela. Cette variable est à changer dans une fiche principale.

L'OBJET

INTRODUCTION

La Programmation Orientée Objets permet, avec LAZARUS, de réaliser vos propres Composants, votre savoir-faire. Avec l'Objet et les Composants, elle permet aussi de réaliser des Logiciels complexes. Ce chapitre et les deux chapitres suivants sont complémentaires. Ils doivent être bien compris afin de surpasser les capacités du programmeur.

Dans ce chapitre nous allons définir ce qu'est l'Objet. Cela va permettre de mieux comprendre LAZARUS, donc de sublimer cet outil par l'automatisation DRA. LAZARUS est un outil de Développement Rapide d'Applications, aboutissement de l'utilisation de la programmation orientée Objets.

Si vous ne connaissez pas l'Objet vos Composants sont mal faits. Il existe des règles simples et des astuces permettant de réaliser vos Composants.

Nous allons dans ce chapitre vous parler simplement de l'Objet.
Étoffez vos connaissances avec des livres sur l'analyse Objet, analyse proche de l'humain. Elle nécessite du travail, de la mémoire, de la prise de notes et de la technique. Nous parlons succinctement des relations en Objet dans le chapitre sur les **Logiciels centralisés**.

Chaque Compilateur possède des spécificités ou améliorations, pour faciliter le travail du développeur. Nous vous les expliquons.

LES OBJETS

Un Objet est une entité interagissant avec son environnement. Par exemple votre "clavier" d'ordinateur est un Objet. Ce "clavier" possède des "touches" qui

peuvent aussi être des Objets "touche", si l'Objet touche est complexe dans notre application.

Vous voyez qu'il est possible de définir un Objet "Touche" ou un Objet "Touches". Préférez l'unicité en mettant en tableau votre Objet "Touche" au sein de votre Objet "Clavier". Vous créez un attribut au sein de votre Objet "Touche".

```
type
        TTouche = class
 LTouche : String;
        end;
        TClavier = class
 LTouches : Array [0..6,0..13] of TTouche;
        end;
```

UNE CLASSE

Avec la notion de programmation par Objets, une méthode de programmation proche de l'humain, il convient d'énoncer la notion de classe.

Lors du tout premier exemple on avait déclaré une classe dans le premier chapitre, en créant un formulaire hérité de la classe "TForm". Cette notion de classe est présente dans le FREE PASCAL. Elle a été ajoutée au PASCAL standard.

Ce que l'on a pu nommer jusqu'à présent Objet est, pour FREE PASCAL, une classe d'Objet. Il s'agit donc d'un type FREE PASCAL. L'Objet FREE PASCAL est une instance de classe, plus simplement un exemplaire d'une classe, son utilisation en mémoire avec la possibilité d'être dupliquée.

On peut remarquer que FREE PASCAL définit une classe comme un Objet (type "classe") ou comme un enregistrement (type "record"). L'enregistrement record ne permet que de définir des méthodes. La classe que vous voyez avant pourrait

donc être redéfinie comme type "record". La classe est l'Objet analytique pouvant agir autour de lui.

Une classe accepte l'instruction "with", Vous pouvez utiliser "with" en prenant garde, toutefois, aux variables ou méthodes identiques, dans des entités différentes.

UNE INSTANCE D'OBJET

UML signifie Langage de Modélisation Unifié. C'est une boîte à outils de graphiques, incontournable en Programmation Objet.
Une instance d'Objet en UML est un Objet en mémoire en FREE PASCAL.
L'Objet en UML est une classe d'Objet en FREE PASCAL.

Le PASCAL Objet permet de créer des Instances d'Objets. Une Instance d'Objet c'est un Objet mis dans la mémoire de l'ordinateur, à l'exécution. Une Instance d'Objet peut être dupliquée. Elle est aussi personnalisée grâce à ses variables.

Si votre Classe d'Objet et ses Classes parentes ne contiennent aucune variable, il est nécessaire de se poser la question de leur utilité. Puis-je gagner du temps en créant une unité de fonctions à la place ?

Vous créez votre Objet, qui est instancié une ou n fois en exécution. Par exemple les formulaires prennent beaucoup de place en mémoire donc on les instancie en général une fois.

Les variables simples de votre Objet sont dupliquées et initialisées à zéro, pour chaque Instance créée.

Les attributs et propriétés

Deux Objets "Clavier" peuvent être faits de la même manière, avec une couleur différente. Ce sont toujours deux Objets "Clavier" avec un attribut différent. Un

attribut peut être une couleur, un nombre, du texte, ou bien un autre Objet.

Seulement en FREE PASCAL les attributs, nommés variables, peuvent être enregistrés dans l'EDI comme des propriétés. Ces variables sont alors sauvées dans le fichier "lfm" de l'Objet "TForm", visible en exécution, ou dans le "TDataModule" invisible.

Dès que vous modifiez un formulaire LAZARUS vous utilisez l'Objet. Seulement ces Objets n'ont pas besoin de l'EDI pour charger leurs propriétés à l'exécution. Une partie du Code Source de développement est exécuté afin de renseigner vos Objets.

Au début du livre un formulaire possédait des propriétés permettant d'automatiser facilement le développement. Les propriétés manipulables dans l'"Inspecteur d'Objets" sont du Code Source servant uniquement au développement. Elles vous ont permis de gagner beaucoup de temps dans la création de votre premier Logiciel LAZARUS.

Les développeurs LAZARUS ont soigneusement séparé les unités d'édition de propriétés, vous permettant de renseigner vos formulaires. Les Objets modifiant les propriétés dans l'"Inspecteur d'Objets" ne sont pas inclus dans votre exécutable. Lorsque vous concevrez votre premier Composant, il faudra aussi séparer le Code Source de développement du Code Source exécuté.
Vous ne devez jamais lier dans vos projets directement vers une unité d'enregistrement de Composants, puisqu'elle ne sert qu'à LAZARUS. C'est pour cette raison que des unités, spécifiques à l'enregistrement des Composants, sont à part des unités de Composants.

Le Code Source exécuté peut être séparé de la conception

Un attribut dans LAZARUS

Dans les classes LAZARUS, les attributs se déclarent avant les méthodes. Une méthode est une fonction ou procédure au sein d'une classe.

```
type
      TForm1 = class ( TForm )
 MonBouton : TButton ;
 public
 constructor Create ( AOwner : TComponent ) ;
      end;
```

L'attribut "MonBouton" crée un Objet bouton. L'attribut du type bouton sert à créer un bouton, dans le formulaire TForm1.

Une propriété dans LAZARUS

```
type
      TForm1 = class ( TForm )
 MonBouton : TButton ;
 procedure OnClickMonBouton ( Sender : TObject ) ;
 private
 FClickBouton : Boolean ;
 public
 constructor Create ( AOwner : TComponent ) ;
 property ClickBouton : Boolean read FClickBouton
write FClickBouton default false;
      end;
```

L'événement "OnClick" du bouton "MonBouton" fait appel à la méthode "OnClickMonBouton".

L'attribut privé "FClickBouton" ne peut être lu dans son descendant. Ainsi on accède à cet attribut par sa propriété publique. Ainsi on peut transformer la fiche en Composant plus facilement.

La propriété permet d'accéder aux attributs directement ou indirectement. Vous pourriez créer dans la propriété une fonction appelée getter, et une procédure

appelée setter, permettant de préparer l'attribut avant de l'utiliser.

La propriété "ClickBouton" permet ici de ne pas avoir à définir de getter et setter afin d'accéder au clique de bouton. Ainsi un getter ou setter peuvent être ajoutés ensuite à la propriété. Le Code Source n'a que très peu changé.

On vous montre les différentes options de déclaration de propriétés. FClickBouton doit être initialisé à "false", dans le Constructeur de la fiche. En effet, la déclaration "property" ne permet pas d'initialiser de valeur par défaut. Une déclaration dans une classe n'agit pas.

Les méthodes et événements

Le clavier agit. Il possède alors des méthodes influençant ses attributs ou d'autres Objets. On pourrait créer une méthode d'appui sur une touche quelconque. La méthode est une procédure ou une fonction centralisée dans un Objet. Une méthode modifie les attributs ou variables du programme.

Seulement en FREE PASCAL il y a plus adaptable que la méthode. L'événement permet de renseigner rapidement tout Objet. Ainsi on ne crée pas l'Objet "Clavier" pour récupérer les événements du clavier.

LAZARUS va facilement propager des événements dans le formulaire de votre projet, puis dans chaque Objet de formulaire. Chaque Objet possède alors une méthode captant le clavier. Vous pouvez renseigner les différents événements clavier dans l'"Inspecteur d'Objets". Ces événements sont accessibles aussi, au sein des différents Objets, grâce à des méthodes.

L'événement consiste à transformer une méthode en variable, pour pouvoir, comme les propriétés, transférer facilement un événement à un autre Objet.

Les comportements de l'Objet

L'Objet est, contrairement aux langages procéduraux, plus proche de l'homme que de la machine. Il permet de réaliser des systèmes humains plus facilement. L'Objet étant plus proche des représentations humaines il complexifie au minimum le travail de la machine en l'organisant en Objets. Un Objet possède trois propriétés fondamentales :
L'Héritage
L'Encapsulation
Le Polymorphisme

L'Objet peut être représenté schématiquement grâce à une boîte à outils nommée UML. Il existe des Logiciels Libres permettant de créer des schémas Objets UML, comme STAR UML ou ou un moteur BPM.

L'HÉRITAGE

Les formulaires LAZARUS utilisent l'Objet. Vous voyez une définition d'un type Objet, dans chacune des unités de votre formulaire. Vous avez donc utilisé de l'Objet sans le savoir.

Un type Objet se déclare de cette manière :

```
type
    TForm1 = class ( TForm )
    end;
```

On crée ici une nouvelle classe qui hérite de la classe "TForm". Vous pouvez appuyer sur "Ctrl" ,puis cliquer sur "TForm", pour aller sur la déclaration du type "TForm". Vous ne voyez alors que des propriétés, car "TForm" hérite lui aussi de "TCustomForm", etc.

Notre formulaire se crée donc grâce à l'Objet "TForm". Cet Objet possède la particularité de gérer un fichier "lfm", contenant les Objets visibles dans la conception du formulaire.

Votre formulaire est compris par LAZARUS par le procédé de l'Héritage. Il

n'utilise que rarement toutes les possibilités du type formulaire surchargé.

Vous pourriez donc utiliser un autre type descendant de "TForm" pour votre formulaire. Cependant tout Composant LAZARUS est hérité du Composant le plus proche de son résultat demandé. On peut donc hériter votre formulaire du Composant "TCustomForm" si certaines propriétés gênent. Remontez jusqu'à "TCustomForm" en cliquant sur le type "TForm" avec Ctrl.

Vous pouvez constater que ces deux Objets ne diffèrent que par la visibilité de leurs propriétés, car "TForm" hérite de "TCustomForm".
Vous voyez des déclarations de variables dans l'Objet "TForm". Certaines variables, comme les Objets, sont mises en mémoire par le Constructeur du formulaire.

Vous ne voyez en général pas de Constructeur dans un formulaire descendant de "TForm". En effet LAZARUS met automatiquement en mémoire vos variables de Composants, déclarées grâce à chaque fichier ".lfm" créé. Ce fichier ".lfm", c'est votre formulaire visuellement modifiable, grâce à la souris et à l'"Inspecteur d'Objets".

LA SURCHARGE

La surcharge consiste dans l'Héritage à modifier le comportement d'une méthode surchargée. Par exemple, on peut hériter de l'Objet "TForm", afin de surcharger la méthode "DoClose". Si, dans un Composant héritant de "TForm", cette méthode surchargée contient le comportement spécifique de toutes les fiches qu'on utilise, on peut automatiser la fermeture de ses formulaires. Il y a moins de Code puisque la fermeture est dans un seul Objet.

Dans le paquet manframes du projet MAN FRAMES, la classe TF_CustomFrameWork contient une méthode DoClose permettant notamment de libérer la fiche, afin de gagner en mémoire. À la fin de la méthode vous voyez une réaffectation des valeurs initiales au cas où l'utilisateur veut personnaliser l'utilisation du formulaire, afin de récupérer des valeurs.

Dans le paquet extcomponents de EXTENDED, il existe une fenêtre principale à hériter nommée TF_FormMainIni, pour laquelle la méthode DoClose est utilisée afin de fermer la connexion de l'application.

Le mot clé "virtual"

Par défaut toute méthode déclarée ne peut être surchargée. Une méthode est donc par défaut statique car il est impossible de la surcharger. En effet une méthode statique ne peut qu'être éludée, pas surchargée.

Pour qu'une méthode puisse être modifiée par ses héritières, ajoutez le mot clé "virtual". Si vous scrutez la toute première déclaration de la méthode DoClose vous trouvez ainsi cette Source dans la première déclaration de DoClose :

```
procedure DoClose(var CloseAction: TCloseAction);
virtual;
```

On peut aussi utiliser cette déclaration :

```
procedure DoClose(var CloseAction: TCloseAction);
dynamic;
```

Il est préférable d'utiliser la déclaration "virtual". En effet cette déclaration favorise la vitesse, plus importante que la taille en mémoire avec "dynamic".

Il existe d'autres moyens pour gagner de l'espace. Nous vous les citons dans le chapitre sur les Composants.

Le mot clé "override"

Pour surcharger une méthode virtuelle, ajoutez cette Source dans une classe descendante. Vous pouvez par exemple ajouter ce Code Source dans tout formulaire :

```
procedure DoClose(var CloseAction: TCloseAction);
override;
```

Cette déclaration dans le formulaire est identique à l'affectation de l'événement "OnClose" de votre formulaire, hérité de "TForm".

Seulement créer cette méthode permet de créer un Composant personnalisé, hérité de "TForm".

Il suffit d'appuyer en même temps sur "Ctrl", puis "Shift", puis "C" pour créer le Code Source à automatiser. Cette Source est ainsi créée :

```
procedure TForm1.DoClose(var CloseAction:
TCloseAction);
begin
 inherited DoClose(CloseAction);
end;
```

Ainsi, lorsque tout Objet Propriétaire de "Doclose" appelle cette méthode, on appelle d'abord la méthode "Doclose" du Composant parent, au dessus de toutes, qui peut éventuellement appeler les méthodes ascendantes "DoClose", par le mot clé "inherited".

Lorsque vous surchargez votre méthode, choisissez toujours d'utiliser le mot clé "inherited". Sinon votre programme boucle à l'infini.

Vous pourriez créer un Composant hérité de "TForm", avec vos options de fermeture de formulaires. Ainsi du Code Source ne serait plus dupliqué.

L'ENCAPSULATION

Nous venons de voir, par la surcharge, le procédé de l'Encapsulation. L'Encapsulation permet de cacher des comportements d'un Objet, afin d'en créer un nouveau.

L'Encapsulation existe déjà dans les langages procéduraux. Vous pouvez déjà

réutiliser une fonction ou procédure, en affectant le même nom à la nouvelle fonction ou procédure, dans une nouvelle unité, qui réutilise la fonction ou procédure. Seulement vous utilisez le nom de l'unité pour distinguer les deux fonctions ou procédures.

On pourrait ainsi appeler la fonction ou procédure imbriquée comme l'original. On pourrait donc se tromper facilement de fonction ou procédure identique, en éludant une unité. L'Encapsulation en PASCAL non Objet est donc approximative.
En effet on évite de créer les mêmes noms de fonctions ou procédures, car cela crée des conflits sans forcément que l'on s'en aperçoive.

Au travers des unités et autres librairies, l'Encapsulation est améliorée en Objet.

L'intérêt de l'Encapsulation est subtil et humain. Si vous créez un descendant d'un bouton en surchargeant la méthode "Click", afin de créer un événement centralisé, et que vous utilisez ce bouton dans votre formulaire, vous vous apercevez que vous ne pourrez jamais appeler la méthode "Click" de ses ancêtres.

Autrement dit l'Encapsulation permet de modifier légèrement le comportement d'un Objet afin d'en créer un nouveau, répondant mieux à ce que l'on cherche. On peut hériter du Composant au-dessus, si l'on veut utiliser la nouvelle gestion, éludant l'ancienne, par le nouveau Composant.

Autrement dit, on s'aperçoit ici que la programmation Objet demande plus de mémoire que la programmation procédurale. En effet il est possible d'éluder le Code Source de la méthode encapsulée.

Cependant l'Objet permet de gagner du temps dans le développement, grâce à l'automatisation des systèmes humains, puis informatiques.
L'Objet permet aussi de maintenir plus facilement le Code Source créé. Avec LAZARUS, tout Objet Composant Open Source d'un formulaire est consultable, par simple clic de souris, grâce à l'inspecteur d'Objet.

Une variable peut ne pas être être héritée, mais peut être de nouveau déclarée.

Cependant l'ancienne variable prend alors la place réservée à son pointeur à l'exécution.

Il est même possible que vous ne puissiez empêcher le Constructeur de la variable de mettre en mémoire ses autres variables, s'il s'agit d'une classe.

Quand trop de variables fantômes sont présentes dans l'ancien Composant hérité, demandez-vous s'il est utile de créer un nouveau Composant, héritant d'un des Composants ascendants. En effet les variables fantômes sont en mémoire, en partie, dans chaque instance de classe.

La déclaration "private"

Une variable privée n'est accessible dans aucun descendant. On ne peut accéder à une méthode, ou une variable "private", que dans l'Objet la possédant. Une méthode "private" ne peut pas être surchargée.

Allez sur la déclaration de "TForm" :

```
 TCustomForm = class(TScrollingWinControl)
 private
 FActiveControl: TWinControl;
 procedure SetActiveControl(AWinControl:
TWinControl);
 published
 property ActiveControl: TWinControl read
FActiveControl write SetActiveControl;
  end;
```

Vous voyez, dans les Sources de LAZARUS, que beaucoup de variables sont déclarées en privées. Elles sont tout de même accessibles dans l' "Inspecteur d'Objets", sous un autre nom.

Souvent, il suffit d'enlever la première lettre de la variable, pour la retrouver dans l'"Inspecteur d'Objets". Des déclarations privées sont utilisées dans les

propriétés publiées de vos Composants, grâce aux getters et setters, eux aussi privés, afin d'être visibles dans l'"Inspecteur d'Objets".

La déclaration "private" permet de regrouper les variables, avec les "getters" et les "setters" des propriétés créées. Un "setter" est une affectation d'une variable, contenant généralement son nom, initialisant une partie du Composant. Le "getter" fait de même, en lisant cette fois la variable privée, contenue dans son libellé.

Ainsi le programmeur ne se trompe pas lorsqu'il réutilise la variable. Il utilise la propriété déclarée par "property", définissant d'éventuels "getters" et "setters", permettant d'initialiser le Composant en affectant la variable. Vous voyez des propriétés dans les Composants LAZARUS. Ces propriétés sont en général visibles dans "l'Inspecteur d'objets".

La déclaration "protected"

Une méthode "protected" ne peut être surchargée que dans l'unité de l'Objet et dans les Objets hérités. Surcharger un Composant permet donc d'accéder aux méthodes ou variables protégées, afin de modifier le comportement d'un Composant.

On met dans la zone "protected" les méthodes de fonctionnement du Composant, pouvant être surchargées, afin d'être modifiées dans un Héritage. Si on ne sait pas comment mettre une méthode dans la zone "private" ou "protected" il est préférable de placer sa méthode dans la zone "protected". En effet la réutilisabilité d'un Composant est essentiellement due au nombre suffisants de méthodes protégées.

Le type "TForm1" peut accéder aux méthodes et variables "protected" de "TForm" et de ses ascendants. Par contre ce type ne peut pas accéder aux méthodes et variables protégées des Composants non hérités, en dehors de son unité.

La déclaration "public"

Une méthode "public" peut être appelée dans tout le Logiciel qui l'utilise. Une variable et une méthode publiques sont toujours accessibles. Ainsi un Constructeur est toujours dans la zone "public" de sa classe.

Si vous ne savez pas mettre une méthode en "public" ou en "protected", cherchez si votre méthode est utilisée plutôt par le développeur utilisateur en "public", que par le développeur de Composants en "protected". Le développeur de Composants modifie les facultés de votre Composant.

La déclaration "published"

Une méthode "published" peut être appelée dans tout le Logiciel qui l'utilise. Une propriété et une méthode publiées sont toujours accessibles.

Vous voyez dans vos Composants que les propriétés "published" sont visibles dans l'inspecteur d'Objet de LAZARUS. La déclaration "published", et les propriétés dénommées par "property", permettent d'améliorer le Polymorphisme du PASCAL Objet.

Ce procédé permet de manipuler les propriétés et méthodes "published", sans avoir à connaître leur propriétaire. Vous pouvez réaliser des procédés de lecture indépendants du type d'Objet, grâce à l'unité "PropEdits".

Ainsi une méthode publiée peut être appelée indépendamment de la classe Objet qui l'utilise. Une propriété publiée est sauvegardée dans le formulaire. Cette propriété est accessible indépendamment de la classe Objet qui l'utilise.

Voici comment récupérer une méthode publiée sans connaître sa classe :

```
// Récupère une propriété d'Objet
// aComp_ComponentToSet : Composant cible
// as_Name  : Propriété cible
```

```
// a_ValueToSet  : Valeur à affecter
function fmet_getComponentMethodProperty ( const
aComp_Component : TComponent ; const as_Name :
String ) : TMethod ;
Begin
 if  assigned ( GetPropInfo ( aComp_Component,
as_Name ))
 and  PropIsType  ( aComp_Component, as_Name ,
tkMethod)
 then Result := GetMethodProp  ( aComp_Component,
as_Name );
End ;
```

Cette méthode permet de récupérer toute méthode publiée "as_Name" de tout Composant. Il suffit ensuite de forcer le type de votre événement.

Par exemple :

```
{$mode DELPHI} // Directive de compilation
permettant d'enlever les ^ de pointeurs, Elle est à
placer au début de l'unité

Var MonClick : TNotifyEvent ;
Begin
 MonClick :=TNotifyEvent (
fmet_getComponentMethodProperty ( AComponent ,
'OnClick' ));
 if assigned ( MonClick ) Then
 MonClick (  Acomponent );
End;
```

Cette Source permet d'exécuter tout événement "OnClick" de tout Composant, s'il est publié.

Dans l'inspecteur d'Objet vous pouvez voir aussi des événements. Les événements sont un appel vers une méthode, grâce à une variable de méthode. Le type de la variable méthode permet d'affecter des paramètres à une méthode.

Nous avons déjà utilisé les événements dans un chapitre précédent.

Un événement est une définition en variable d'une méthode. Il est cependant possible d'accéder à une méthode publiée, sans avoir à utiliser l'événement, en utilisant le type "TMethod". Ce type permet d'accéder à une méthode publiée dans un Composant.

```
{$Mode DELPHI}
var lmet_MethodeDistribueeSearch: TMethod;
Begin
 lmet_MethodeDistribueeSearch.Data := Self ;
 lmet_MethodeDistribueeSearch.Code :=
MethodAddress('MaMethodePubliee') ;
 ( lmet_MethodeDistribueeSearch as MonTypeMethod
) ( monparametre ) ;
End;
```

Si ce que vous avez défini n'existe pas déjà on définit un type méthode ainsi :

```
        TMaMethode        =        procedure(const
monparametre:TypeParametre)
 of object;
```

Le fait de noter "of object" indique que l'événement est dans un objet, ceci afin de mieux l'éditer. Si vous éludez le "of object" vous pouvez utiliser votre déclaration pour affecter directement une procédure ou fonction à une variable, ou même une constante.

L'exemple est dans le paquet IBEXPRESS pour LAZARUS :

```
 type
   TOnGetLibraryName   =   procedure(var   libname:
string);

const
 OnGetLibraryName: TOnGetLibraryName = nil;
...
initialization
      OnGetLibraryName:=       TOnGetLibraryName(
p_setLibrary);
end.
```

Les événements automatisent la gestion de méthodes publiées. Ainsi un événement devient une adresse manipulable grâce aux types d'événements, décrits ci-dessus.
Il est ainsi possible de créer ceci dans un Objet :

```
Property OnMaMethode : TMaMethode read
FMaMethode write FMaMethode ;
```

Il est nécessaire de placer ceci dans le constructeur de l'Objet :

```
FMaMethode:= nil ;
```

Vous avez une utilisation des méthodes publiées dans les projets MAN FRAMES et XML FRAMES, dans la fenêtre principale, liée à fonctions_Objets*. N'hésitez pas à faire une recherche dans le répertoire du paquet, afin de trouver l'utilisation de la méthode publiée.

La déclaration "published"

Lorsqu'on publie une propriété dans un Composant référencé celle-ci se retrouve dans l'inspecteur d'Objet.

Il est préférable de créer une propriété au plus proche de l'Objet, en surcharge, voire en participation Libre, si la modification est utile pour la communauté.

Voici un bouton amélioré pour une utilisation particulière :

```
type
    TMyButton = class ( TSpeedButton )
 private
 FClickBouton : Boolean ;
 function getClickBouton:Boolean;
 public
 procedure Click ; override ;
 published
 property ClickButton : Boolean read
getClickBouton write FClickBouton default false ;
 property Glyph stored false ;
```

```
end;
```

La propriété "ClickButton" permet de savoir si le bouton a été cliqué.
Le getter getClickBouton permet de réinitialiser FClickBouton à la lecture du clique.

On empêche ici d'enregistrer l'image du bouton, en surchargeant la déclaration publiée du "TSpeedButton". Cela permet d'empêcher d'enregistrer dans chaque fiche l'image qu'on aurait chargée. Nous vous expliquons cette optimisation dans le chapitre sur la création d'un Composant.

L'option "stored" à "false" n'enregistre pas la propriété dans l'"Inspecteur d'Objets", donc dans l'exécutable. "stored", qui est à "true" par défaut, permet l'enregistrement dans l'"Inspecteur d'Objets", si votre Composant est dans la palette des Composants.

Les Constructeurs et Destructeurs

Les variables simples, comme on l'a vu, se libèrent automatiquement. FREE PASCAL ne libère pas automatiquement certaines variables, notamment les Classes d'Objets utilisées. Il faut soit détruire soi-même soit utiliser des Composants se libérant correctement.

Un descendant de TComponent détruit automatiquement les Objets de type TComponent, qui lui ont été affectés. Donc, si vous utilisez un descendant de TComponent dans votre classe, il suffit, pour le détruire automatiquement, d'affecter le Propriétaire du Composant, comme étant votre classe d'Objet.

Tout descendant de TComponent peut être enregistré dans l'"Inspecteur d'Objets".

Pour les autres classes d'Objet, il est nécessaire d'utiliser les Constructeurs et Destructeurs, pour d'abord les initialiser, enfin les libérer.

Voici un exemple de Constructeur et de Destructeur :

```
Interface
uses Classes ;

 TReadText = class(TComponent)
 MonFichierTexte : TStringlist;
 public
 constructor Create(TheOwner : TComponent);
 destructor Destroy;
 end ;

implementation

constructor        TReadText.Create(TheOwner        :
TComponent);
begin
  Inherited Create(TheOwner);
 MonFichierTexte := nil;
end ;
destructor TReadText.Destroy;
begin
  Inherited Create(TheOwner);
 MonFichierTexte.Free;
end;
```

Tout Objet défini dans une classe doit être initialisé à "nil". Le Constructeur initialise le "TStringList". Il peut alors être créé dès qu'on l'utilise, grâce à une méthode centralisée.

Le Destructeur du Composant libère les Objets mis en mémoire. Ainsi la méthode statique "Free", de la classe "TObject", vérifie si la variable est à "nil". Puis, si elle n'est pas à "nil", elle appelle le Destructeur de l'Objet de type "TStringList".

Vous avez en général, dans beaucoup de Composants LAZARUS, un constructeur et un destructeur surchargés.

Le Polymorphisme

Le terme Polymorphisme est certainement le plus redouté. Pour le comprendre, voici la sémantique du mot : "poly" signifie plusieurs et "morphisme" signifie forme. Le Polymorphisme traite de la capacité de l'Objet à posséder plusieurs formes.

Cette capacité dérive directement du principe d'Héritage, vu précédemment. En effet, un Objet hérite des champs et méthodes de ses ancêtres. On peut ainsi redéfinir une méthode, afin de la réécrire ou de la compléter.

On considère un Objet Véhicule et ses descendants Bateau, Avion, Voiture. Ces Objets possèdent tous une méthode Avancer, le système appelle la fonction Avancer spécifique, suivant que le véhicule est un Bateau, un Avion ou bien une Voiture. Ainsi la méthode avancer sera unique, pour 3 Objets différents.

Le concept de Polymorphisme permet de choisir, en fonction des besoins, quelle méthode ancêtre appeler. Le comportement d'un Objet polymorphe devient donc modifiable à volonté.

Le Polymorphisme est donc la capacité du système à choisir dynamiquement la méthode de l'Objet en cours.

Attention !

Le concept de Polymorphisme en orienté Objet ne doit pas être confondu avec celui d'Héritage multiple en Objet pur. L'Héritage multiple est interdit par le FREE PASCAL. Il permet à un Objet d'hériter des champs et méthodes de plusieurs Objets à la fois. Le Polymorphisme permet de modifier le comportement d'un Objet et celui de ses descendants au cours de l'exécution.

L'Héritage multiple possède lui aussi ses défauts. Il est encore plus lourd en mémoire que le Polymorphisme en orienté Objet. L'Héritage multiple nécessite plus de rigueur en amont.

L'Abstraction

L'Abstraction ne sert, en FREE PASCAL, qu'à centraliser plusieurs Objets, en héritant d'un Objet abstrait donc pas utilisable. Cet Objet abstrait possède en fait des méthodes dites "abstraites" ou "abstract", non implémentées dans l'Objet abstrait.

Voici à quoi l'Abstraction ressemble dans une classe d'Objet :
procedure Nager; **abstract;**

Cette déclaration est la seule à ne pas demander d'implémentation. Elle crée une erreur malgré tout, à l'exécution, si son descendant n'implémente pas la méthode "Nager".

Ainsi l'Objet abstrait qui nage n'a pas besoin de savoir comment il nage. Les différents descendants nagent en fonction du type classe créé. Une classe "Poisson" nage en oscillations horizontale. Une classe "Humain" nage en brasse, en crawle, en papillon, etc.

On crée donc dans la classe Humain des méthodes secondaires. On crée dans une unité de fonctions les différentes méthodes à réutiliser.

Il reste à définir comment on choisit le type de nage en fonction de l'Objet créé. On peut définir le type de nage grâce à une énumération en paramètre.

Le type "interface"

Le type "interface" est le Polymorphisme en FREE PASCAL en son essence.

Il est impossible d'hériter de deux classes voire plus. Utilisez le type "interface" pour remédier à ce problème. Une classe FREE PASCAL hérite d'une classe, avec, éventuellement, un ensemble d'interfaces.

Le type "interface" permet de créer des Objets polymorphes. Il peut s'ajouter à un Héritage. On crée le type interface comme si on crée une classe. Cependant il n'est possible que d'y ajouter des méthodes abstraites.

Nous parlons du type "interface" dans le chapitre sur les Composants.

Polymorphe avec les propriétés publiées

Nous avons vu, précédemment, qu'il est possible d'appeler n'importe quel événement "Onclick", de n'importe quel Composant.

Le type classe ou Objet "TForm", c'est un Composant particulier. Le Composant et Objet "TForm" permettent de créer d'autres Objets Composants, à l'aide d'un fichier avec l'extension ".lfm". Vous pouvez donc changer de Composants dans vos formulaires, en changeant le type de votre Composant Source vers votre type destination. Changez alors le type de votre Composant, à la fois dans le fichier ".pas" de votre ficher, puis dans le fichier ".lfm".

Tout d'abord allez sur le formulaire à transformer.
Ajoutez un Composant "TLabel". Redimensionnez-le.

Allez sur la déclaration classe de sa formulaire dans le fichier ".pas".
Changez le type de son Composant vers le nouveau type de Composant, sans vous tromper. Vous pouvez par exemple lui affecter le type "TButton".

Allez sur le formulaire visuel et cliquez sur le bouton droit dessus. Choisissez "Afficher le Source".
Retrouvez votre Composant en faisant une recherche. Puis changez le type par le même type "TButton".

Fermez et rouvrez le formulaire. Des propriétés n'existent plus. LAZARUS demande de les effacer.

Votre nouveau Composant "TButton" a alors remplacé l'ancien, avec les propriétés identiques de ce dernier.
C'est la nomenclature LAZARUS qui a permis d'adapter les propriétés anciennes, vers le nouveau Composant.

Autrement dit, avant de créer une nouvelle propriété LAZARUS, il est nécessaire de connaître les propriétés standards de tout Objet LAZARUS. Une propriété LAZARUS est en anglais.

Les types "interface" ne permettent pas de déclarer de véritables Objets. Ils ne font que centraliser la déclaration de méthodes. Les unités de fonctions permettent de centraliser le Code pour le type "interface".

La déclaration "published" et les propriétés permettent d'accéder à d'autres Composants devenus publics et publiés, sans avoir à connaître leur type "class" ou "interface".

La déclaration "published" et les unités de fonctions résolvent partiellement le Polymorphisme. Elles permettent cependant une hiérarchisation de vos Sources, grâce aux paquets.

LES PROPRIÉTÉS

FREE PASCAL possède une syntaxe permettant de créer des Composants, qui sont des Objets avec des propriétés publiées. Les propriétés permettent, si elles sont correctement créées, de mieux gérer le Polymorphisme. Les propriétés publiées permettent la rapidité de conception, en étant présentes dans l'"Inspecteur d'Objets".

Il ne faut pas être créatif pour nommer ses propriétés. Reprenez ce qui existe déjà au sein de LAZARUS, afin de mieux gérer le Polymorphisme. Une propriété doit être nommée du même nom que les propriétés de même nature.
Il est donc nécessaire de connaître les noms des propriétés des Composants de

même nature, afin de créer des propriétés identiques.

L'UML ou le BPM pour programmer en Objets

Des outils UML ou BPM Libres existent pour automatiser la création de vos Logiciels orientés Objets.
UML en anglais signifie Unified Modeling Language, pour Langage de Modélisation Unifié. Ainsi tout outil UML peut être compatible avec d'autres outils UML. Vérifiez cependant si c'est bien le cas.

BPM signifie Business Process Model, pour Modèle de Processus de Gestion. Un modèle BPM permet de créer vite un logiciel de gestion, grâce notamment à XML Frames.

STAR UML

STAR UML est un outil Libre donc partagé en licence GNU/GPL.

Il a été fait sous DELPHI. Vous pouvez donc tenter de le migrer vers LAZARUS après avoir vérifié que quelqu'un ne le fait pas déjà. Dans ce cas, participez au projet de migration, qui peut être intégré au projet STAR UML.

CRÉER SON SAVOIR-FAIRE

INTRODUCTION

La communauté PASCAL a toujours essayé de préparer au maximum le travail du programmeur. Ainsi le programmeur de Composants LAZARUS s'y retrouvera mieux sur d'autres outils de Programmation Objet, ayant connu la simplicité et l'efficacité des Composants de DRA (Développement Rapide d'Applications) en Objet.

Pour créer un savoir-faire respectez un seul principe : Évitez le copier-coller.
On crée donc avec cette technique un savoir-faire :
Au début il n'y a que des unités de fonctions
Puis on utilise et surcharge des Composants
On crée des paquets de Composants
On ouvre alors sa librairie aux autres API
On automatise les paquets en une librairie
On crée une Intelligence Artificielle automatisant les choix
La librairie nécessite peu de Code à créer ou aucun
On crée le reste du code automatiquement

CRÉER DES UNITÉS DE FONCTIONS

Vous pouvez vous aussi créer votre savoir-faire en créant d'abord des projets. Pour créer ces projets, nous vous apprenons à centraliser ce qui aurait été redondant. Évitez le copier-coller. Vous créez des unités de fonctions en centralisant un Code redondant. Vérifiez si votre fonction n'existe pas déjà en cherchant en anglais sur le web.

Une unité de fonction c'est un regroupement de fonctions autour d'un même thème. Si la fonction que vous ajoutez n'est pas immédiatement associée à ce

thème, n'hésitez pas à trouver un nouveau thème. On crée alors une nouvelle unité.

LES COMPOSANTS

LAZARUS sans les Composants ne serait pas utile. LAZARUS permet de créer rapidement des Logiciels grâce aux Composants. La programmation par Composants est rapide sur LAZARUS grâce à l'"Inspecteur d'Objets". Vous pouvez décupler votre vitesse de création avec LAZARUS.

Il est recommandé d'améliorer ou de créer des Composants, qui sont facilement mis en place. Ainsi votre savoir-faire ou vos Composants sont centralisés dans des paquets de Composants, puis dans une voire plusieurs librairies utilisant vos Composants. La librairie contient en plus des outils préparant le travail.

Un paquet est un regroupement de Composants. Les paquets sont installés rapidement sur LAZARUS. Les Composants sont visibles dans LAZARUS. Les Composants de DRA sont facilement modifiables s'il sont fournis avec leurs Sources. Il s'agit cependant d'être expérimenté en PASCAL Objet.

Vos unités de fonctions peuvent ensuite améliorer des Composants par le procédé de l'Héritage, ou par la participation Open Source. Vous créez alors votre premier paquet, en apprenant l'Objet. Vos unités de fonctions sont utilisées et améliorées.

Vous participez alors à un projet Open Source après avoir vérifié la licence Open Source. Si la participation au projet est refusée il est nécessaire de s'interpeller sur l'utilité de cette participation. L'entreprise qui refuse la participation peut avoir d'autres objectifs que les vôtres. Vous pouvez alors surcharger le Composant.

Un Composant LAZARUS est mis en place rapidement grâce à l'"Inspecteur d'Objets". Cet inspecteur permet d'affecter les propriétés de vos Composants, comme si vous étiez un simple utilisateur. Les propriétés permettant de développer sont un supplément au formulaire ouvert. Elles sont et doivent donc être situées en dehors du Code Source d'exécution.

Votre Composant peut être amélioré grâce à la communauté. Cela ne vous empêche pas, par contre, de toujours être compétitif en recherchant d'autres projets à créer, pour que vos ou d'autres Composants en héritent.

L'économie de travail

À l'heure actuelle nos libertés et nos droits dépendent de l'économie de travail réalisée avec la production industrielle et agroalimentaire, grâce aux énergies de plus en plus denses. Par exemple le charbon est moins dense que le pétrole. Le pétrole permet donc de faire plus de choses. Le thorium puis l'helium 3 permettront d'en faire encore plus.

Le métier d'informaticien consiste à optimiser les industries. Il consiste donc aussi à optimiser son travail. Le RAD pour Rapid Application Development, et son optimisation le VRAD servent à cela.

Le Composant UML

Les Composants de DRA sont dans une palette et peuvent être facilement inclus dans vos sources. Un Composant UML est un regroupement d'objets créant un micro-système remplissant une tâche définie. Le Composant RAD (version anglaise du DRA) est plus facile à comprendre, puisqu'il est visible. Sachez seulement que le Composant est l'ensemble de votre unité et un peu plus.

Le Composant UML est plus abstrait que le Composant LAZARUS. Le Composant LAZARUS correspond cependant à la version UML. Peu de langages intègrent les Composants, plus proches de la vision humaine d'un système, permettant donc de développer mieux et plus vite.

Son savoir-faire

On crée des unités de fonctions, puis des Composants regroupés en paquets LAZARUS, puis une voire plusieurs librairies. Ces librairies sont indépendantes de LAZARUS, en étant entièrement automatisées grâce à des fichiers.

L'aboutissement est la prise en compte de ce qui est demandé par les clients, en créant des fichiers automatisant la librairie. Ces fichiers sont dédiés à ce qui est demandé. Ce sont les fichiers métiers. Ces fichiers peuvent être automatiquement créés par l'analyste.

Il y a alors plus de similitudes entre l'analyse et le Logiciel, puisque l'on peut passer plus de temps à analyser. En effet il est difficile d'avoir une analyse identique au Logiciel demandé, sans service qualité ou sans automatisation.

Surcharger son savoir-faire afin que l'analyse crée le Logiciel sera la finalisation de l'ensemble des étapes d'automatisation. Vous pourrez alors créer le Logiciel avec le client.

Une fois que ses Composants sont stables, on peut les ajouter au gestionnaire de paquets en ligne, en modifiant les préférences du gestionnaire.

Développement Très Rapide d'Applications

Le Développement Très Rapide d'Applications c'est l'aboutissement de tout savoir-faire de DRA. L'utilisation du mot "Très" signifie que l'on crée le Logiciel directement, à partir de l'analyse du projet. On utilise la Programmation Orientée Objet afin de séparer ce qui est demandé par le client, du Logiciel en lui-même.

On supprime alors une étape de programmation, à savoir la transcription de l'analyse en Logiciel, car celle-ci est automatisée par la Programmation Objet et les fichiers passifs. Les fichiers passifs contenant une analyse du Logiciel sont lus par le moteur de DTRA, afin de créer le Logiciel.

Un Logiciel créé en Développement Très Rapide est toujours conforme à

l'analyse, car l'analyse crée le Logiciel. En effet ce que demande le client est une vision humaine pouvant être comprise par un moteur DTRA. Des méta-modèles existent, permettant de synthétiser cette vision humaine.

CRÉER UN FRAMEWORK DTRA

Le DTRA est une branche de l'Ingénierie Pilotée par les Modèles, Model Driven Engineering on MDE en anglais.

Pour trouver un Framework DTRA, cherchez "bpmn engines". Le format BPMN 2, qui répertorie le format des données ainsi que le format des variables calculées, permet de transférer les applications entre les Frameworks DTRA. Le format BPEL permet de répertorier les actions sur le format BPMN 2.

Il existe des Frameworks ou savoir-faire DTRA JAVA permettant de créer des Logiciels de gestion avec fiabilité. MICROSOFT a aussi créé des méta-modèles analytiques.

W4 EXPRESS utilise des fichiers de description des méta-données. Ces informations en langage XML et leurs descriptions servent à définir comment on utilise une quelconque donnée. Cela permet d'homogénéiser les Logiciels pour pouvoir les créer plus vite. W4 EXPRESS n'est plus mis à jour. Cependant il est utilisé dans XML FRAMES. XML FFRAMES est un savoir-faire Client/Serveur, qui crée des applications de gestion sur WINDOWS, GNOME, KDE, voire MACOSX.

Il est possible de créer son Framework Web de Développement Très Rapide, si la création de son Logiciel est répétitive. Ainsi l'automatisation permet de gagner du temps, ensuite.

Un Logiciel de gestion c'est un Logiciel gérant des processus. Il est composé d'un lien vers les données, de fiches, de relations, de filtres, de statistiques, de feuilles de calcul, etc. Il est simple à modéliser donc simple à automatiser.

Les Composants ou plugins permettent de créer une partie du Logiciel selon les spécificités demandées. Ces Composants ou plugins permettent d'étendre les capacités techniques du moteur. Ils nécessitent une approche à la fois élémentaire, pour répondre à une partie infime de la demande, puis globale pour être inclus dans toute analyse. Ils sont utilisés dans l'EDI de DTRA ou EDTRA.

Il suffit alors d'utiliser une Ingénierie Pilotée par Modèle, afin de centraliser ce genre de Logiciel dans un moteur contenant les Composants. Les Composants sont renseignés par les modèles analytiques, pouvant lire toute analyse. Les Logiciels, créés sans le moteur DTRA, renseignent uniquement les Composants.

Ainsi il y a au moins deux couches dans un modèle DTRA, à savoir les Composants, puis le moteur qui renseigne les Composants. Il s'agit donc de EXTENDED et XML FRAMES.

Une troisième couche intermédiaire est représentée par les composants qui permettent de se passer des fichiers analytique, mais qui ne servent pas pour une application standard. Il s'agit de MAN FRAMES. Cette couche contient la partie création de données.

Une fois cette étape passée les Logiciels peuvent être en partie traduits, par le reverse engineering traduisant les données en un début de Logiciel. Vous créez ensuite la partie dynamique, non incluse dans les données.

On peut créer une quatrième couche, qui va permettre d'analyser ce que veut l'utilisateur, par une Intelligence Artificielle.

Il est donc possible de créer des plugins, dans les Frameworks ATK ou JELIX JFORMS, ou tout Framework de gestion, afin de permettre leur automatisation.

Intérêts du DTRA

Mis à part le gain de temps, supprimer une étape de programmation permet

d'améliorer aussi la qualité des Logiciels créés. L'analyse correspond toujours au Logiciel, puisque l'analyse crée le Logiciel. Les jeux de tests sont mis en place pour le moteur DTRA uniquement.

Même si un moteur DTRA nécessite au moins un ingénieur développeur, ce dernier pense fonctionnalités et pérennité du système.

Les fichiers passifs peuvent être lus par d'autres moteurs. XML FRAMES utilise les fichiers passifs de W4 EXPRESS. Il est possible de créer des Forks de W4 EXPRESS sur d'autres langages, grâce à des Frameworks de gestion, avec fiches et relations.

Créer un Composant

Nous allons ajouter un fonctionnement automatisé à un bouton. Ce Composant surchargé fermera sa fenêtre. Nous allons aussi lui affecter une image. Pour tout bouton créé, sera affectée une seule image. Ainsi l'exécutable est plus léger. Ce bouton permet de fermer un formulaire.

Le Composant qui va être créé va donc répondre à une demande technique : La taille de l'exécutable et la centralisation des Sources. La centralisation des Sources va permettre de répondre à l'automatisation de son savoir-faire. Cela va donc permettre de répondre plus rapidement à la demande du client.

Pour créer votre paquet personnalisé Faites "Fichier" puis "Nouveau" puis "Paquet".

Créer un nouveau paquet dans l'"EDI LAZARUS"

Nommez le paquet "LightButtons".

"Ajoutez" la condition "LCL". Une condition est un paquet utilisé. Le paquet LCL contient tous les Composants visuels standards.

Si vous voulez gagner du temps sur la création de votre composant, vous pouvez "créer un Composant" dans le menu du paquet. Nous vous expliquons comment faire sans passer par cette aide ensuite.

"Ajoutez" un "Nouveau fichier" "Unité", puis sauvegardez votre unité sous le nom "u_buttons_appli". Faites attention à ne pas affecter un nom de fichier identique à une unité existante, car LAZARUS n'aime pas du tout cela. Il veut que tout fichier portant le même nom soit exactement identique.

Enregistrez votre paquet en cliquant sur "Enregistrer", dans le projet de paquet. Affectez lui le nom "LazBoutonsPersonnalises".

Ajoutez ce type dans la partie interface de votre unité :

```
{$mode DELPHI}
interface
uses  StdCtrls, Classes, lresources ;

type
 IMyButton = interface
 ['{620FE27F-98C1-4A6D-E54F-FE57A06207D5}']
 End ;
```

La source placée après interface s'appelle le GUID. Il s'agit de l'identifiant unique permettant de retrouver plus facilement votre interface.

Il est inutile de recopier entièrement cette partie. Créez votre GUID en ajoutant une ligne après interface, puis en appuyant simultanément sur Shift+Ctrl+G.

L'unité "StdCtrls" contient les boutons. L'unité "Classes" contient le type Composant. L'unité "lresources" vous permettra d'ajouter les images à vos boutons.

On utilise le mode DELPHI afin de ne pas avoir à gérer les adresses de pointeurs. Le mode DELPHI est expliqué dans le chapitre **De PASCAL vers FREE PASCAL**.

On déclare un type interface permettant d'indiquer que nos boutons supportent tous le type "IMyButton".

La chaîne hexadécimale permet d'utiliser le comparateur "is" dans toute classe afin de reconnaître le type "IMyButton". Ainsi nous retrouvons plus facilement nos boutons. Nous utilisons le Polymorphisme pour reconnaître nos propres boutons.

Voici le Code Source permettant de reconnaître nos boutons :

```
If UneClasse is IMyButton Then
 Begin
 ShowMessage ( 'C'est mon bouton !' );
  End;
```

En FREE PASCAL on définit au maximum ce que l'on fait. Les définitions d'interface permettent d'utiliser l'Objet à ses possibilités maximum afin d'améliorer les interfaces humaines, en améliorant l'utilisation de ses Composants.

Juste après la définition de l'interface "IMyButton" placez ce Code :

```
 TMyClose = class ( TBitBtn,IMyButton )
 private
 public
    constructor    Create(TheOwner:    TComponent);
override;
 procedure Click; override;
 published
 property Glyph stored False;
 End;
```

TMyClose est le descendant de TBitBtn. Il supporte l'interface IMyButton.

Le Constructeur Create et la méthode Click sont présents dans le Composant TBitButton par l'Héritage. Nous les surchargeons par le mot clé "override" suivi d'un ";".

Grâce au Polymorphisme LAZARUS vous pouvez changer l'ancêtre "TBitBtn" par un ancêtre bouton possédant un "Glyph". D'autres boutons avec "Glyph" existent en dehors du projet LAZARUS. Vous pouvez donc facilement changer de type Bouton par la classe que vous créez.

Tous les Composants LAZARUS possèdent le Constructeur Create avec, comme paramètre, le Propriétaire du Composant. Notre Composant est donc automatiquement détruit lorsque son parent se détruit.

Aussi tous les Composants descendants de la classe TLCLComponent possèdent la méthode "Click" sans aucun paramètre. Une méthode qui peut être surchargée possède le mot clé "virtual".

Le Composant ancêtre définissant la méthode utilise donc le mot clé "virtual" sur la méthode créée. Vous pouvez donc facilement vérifier si c'est bien TLCLComponent qui définit "Click" en maintenant "Ctrl" enfoncée puis en cliquant sur la procédure Click jusqu'à ne plus trouver override.

Pour chercher l'unité du bouton "TBitBtn" cherchez dans tous les répertoires du Code Source de LAZARUS la bonne unité. Sinon cette unité s'appelle "StdCtrls". Cette unité est incluse dans le paquet LCL, la Librairie de Composants visuels.

Dans le paquet "Ajoutez" la "Condition" "LCL".

Ajouter la condition "LCL" au paquet

Vous pouvez "Compiler" le paquet pour trouver des erreurs. Si votre Code Source est bon vous pouvez appuyez sur "Ctrl"+ "Shift" + "C". Cela va créer les deux méthodes.

Maintenant nous allons remplir les deux méthodes surchargées dans la partie "implementation" :

```
implementation

uses Forms;

  // Créer des constantes permet d'éviter de se
tromper  pour leur réutilisation
const
  CST_FWCLOSE='TMyClose';   // Nom du fichier
```

Image

```
{ TMyClose }
procedure TMyClose.Click;
begin
 if not assigned ( OnClick )
 and ( Owner is TCustomForm ) then
 Begin
 ( Owner as TCustomForm ).Close; // Fermeture du formulaire
 Exit;
 End;
 inherited;
end;

constructor TMyClose.Create(TheOwner: TComponent);
begin
 inherited Create ( TheOwner );
 if Glyph.Empty then
 Begin
 // Charge le fichier Image
   Glyph.LoadFromLazarusResource ( CST_MYCLOSE);
 End;
end;
```

Il est nécessaire d'utiliser des constantes classées. Elles permettent de centraliser les chaînes afin d'optimiser. Ajoutez la constante CST_MYCLOSE à la partie interface de votre unité de Composant. Affectez lui le nom de votre classe créée, à savoir "TMyClose".

Voilà l'essentiel de notre Composant "TMyClose". La procédure surchargée "Click" ferme le formulaire, s'il n'y a pas d'événement de "Click" sur le Bouton dans le formulaire. Il suffit que cette Source soit exécutée deux fois dans l'Application pour que cette Source ait remplit un objectif d'automatisation.

On passe la méthode "Click" héritée par la méthode "Exit". On a vérifié, avant d'éluder la méthode "Click" de l'ancêtre, qu'il n'y avait que la gestion de l'événement "OnClick" que l'on évitait. Nous n'éludons effectivement pas

l'événement "OnClick" en vérifiant s'il existe. Ainsi nous permettons d'utiliser le bouton de fermeture uniquement pour l'image centralisée.

Le Code spécifique est automatisé avec le maximum de Composants possédant différents objectifs. On se rend compte que, bien que le Composant permette de centraliser, celui-ci augmente aussi la taille de l'exécutable.

Si la fonction de fermeture était sur chaque événement de chaque bouton, cela alourdirait l'exécutable. Aussi, si les conditions techniques de fermeture du formulaire changeaient, il faudrait changer le Code Source de chaque bouton ne descendant pas de "TMyClose".

Choix de l'image

Le Code Source du Constructeur va créer une erreur à l'exécution car il n'y a pas de fichier image.

Le Constructeur de notre bouton charge l'image du bouton grâce au fichier "lrs".

Choisissez votre image de fermeture sur un site Web comme OPEN CLIPART, contenant des images Libres avec une licence "Domaine Public". Une licence "Domaine Public" est utilisable en citant uniquement la source, afin de se protéger. C'est la licence la plus libre qui soit. Il existe aussi dans le même genre la CC0, la Creative Common sans rien du tout, la licence BSD aussi.

Il existe aussi des images avec des licences "Art Libre", ou "Creative Common" avec ou sans "by" avec ou sans "SA". Le "by" signifie qu'on doit citer l'auteur.

Vérifiez la licence. Si vous vous posez des questions sur les licences vous pouvez contacter un Groupe d'Utilisateurs LINUX local ou GUL.

Les licences Libres possèdent plus de facilités quant aux modifications. Vous avez une définition de Libre dans le glossaire.

Le fichier image doit être un fichier "XPM", de 24 pixels sur 24 pixels, avec comme nom le type de la classe de notre bouton. Cela permet de charger l'image du bouton dans la palette de Composants. L'image peut être transparente en partie, afin de l'intégrer plus facilement.

Convertissez avec GIMP votre image au format "XPM". Pour faire cela sauvegardez l'image avec GIMP, en remplaçant l'ancienne extension de fichier par "XPM" sans enlever le point. L'extension ce sont les dernières lettres après le dernier point du fichier.

Créez dans le répertoire du paquet un fichier vide "u_buttons_appli.lrs". Allez dans votre dossier contenant "LAZARUS". Allez dans le répertoire "tools".

Compilez le projet "glazres", si ce n'est pas déjà fait. Avec vous créez votre fichier "lrs" à partir d'images.

Vous pouvez éventuellement ajouter d'autres images, si vous voulez créer d'autres boutons.

Nous allons ajouter le fichier ressource à l'unité.

Allez tout à la fin de l'unité de votre Composant. Insérez cette Source avant le "end." final :

```
initialization
 {$I u_buttons_appli.lrs}
end.
```

L'unité "lresources" est nécessaire pour les fichiers lrs. Vous pouvez aussi utiliser les fichiers ".Res" de DELPHI.
La directive {$I} ajoute le fichier ressource au Code Source.

Enregistrement du Composant

Enregistrer le Composant permet de voir certaines de ses propriétés reconnues par LAZARUS, afin d'éditer rapidement le Composant.

Il est possible d'utiliser des propriétés, voire de les surcharger, dans son unité d'enregistrement de Composants.

Un Composant est un Objet descendant de l'Objet TComponent. Tout Composant LAZARUS, descendant de TComponent, peut être accessible dans la palette de Composant.

Pour qu'un Composant soit enregistré dans une palette, voici le Code Source à affecter à une unité d'enregistrement du Composant.

Créez cette nouvelle unité dans votre nouveau paquet :

```
Unit u_regboutons;

interface

uses Classes;

procedure Register;

implementation

uses u_buttons_appli;

procedure Register;
begin
 RegisterComponents('MesBoutons', [TMyClose]);
end;

end.
```

La procédure "Register" doit être exécutée par le paquet qui installe le Composant. Vérifiez pour cette unité si la case à cocher "Référencer l'unité" est cochée.

L'unité d'enregistrement ne doit jamais être utilisée dans votre programme. En effet le Code Source de développement est inutile pour le programme exécutable.

"Référencer l'unité" est coché pour l'unité

Modifier ou Surcharger ?

Si votre Composant ajoute un objectif au Composant initial, il est préférable de créer une autre unité surchargeant le Composant. Cette dernière permet de ne pas avoir de Code inutilisé pour le Composant à surcharger.

La modification d'un Composant se fait en deux étapes. Pour pouvoir modifier un Composant, l'accord de l'auteur est nécessaire. Vous avez alors accès au gestionnaire de version permettant de le modifier.

Sinon le Composant peut être abandonné, et sa licence permet de le modifier. S'il n'y a aucune licence, contactez l'auteur.

Sinon vous ne faites qu'ajouter des options mineures ou vous créez un Fork, un second projet évoluant différemment. Seulement, sans l'accord de participation de l'auteur, vous ne pouvez pas correctement mettre à jour le Composant, Vous êtes alors seul responsable de ses évolutions. Il suffit souvent de contacter l'auteur, qui sera en général content de vous aider à améliorer son savoir-faire.

Surcharger un Composant

N'hésitez pas à surcharger tout Composant, surtout si vous disposez des Sources qui permettent de trouver plus facilement les contournements pour les erreurs d'Héritage.

Lorsque vous surchargez un Composant, vous avez accès aux méthodes protégées. Ces méthodes sont inaccessibles sans la surcharge.

Il faut quelquefois être astucieux, afin de surcharger correctement certaines méthodes, de certains Composants. Vous pouvez indiquer à l'auteur du Composant les modifications nécessaires à un Héritage facilement effectué. Évitez d'utiliser des contournements, car ils peuvent provoquer de nouvelles erreurs à la mise à jour.

Créer une librairie homogène

Un paquet de Composants est mis en place rapidement. En voulant être encore plus rapide, vos paquets deviennent une librairie complète paramétrable au minimum voulu. Vous sublimez alors votre capacité d'adaptation en ne créant que l'utile.

Ce qui sera personnalisé sera paramétrable. Le reste se configurera automatiquement.

Une librairie fournit des outils et des procédés pour créer son Logiciel sans perdre de temps. Il faudra améliorer l'EDI LAZARUS ou bien éviter d'utiliser l'EDI avec des fichiers de configuration. Ces fichiers pourront être idéalement des fichiers analytiques UML ou MERISE. Ces fichiers vous permettront d'être indépendant de LAZARUS, en utilisant les mêmes savoir-faire sur d'autres outils de programmation.

Le Libre et l'entreprise

LAZARUS contredit les professeurs ou institutions, indiquant que les Composants nécessitent plus de temps en maintenance qu'en gain de temps à l'utilisation. Un Composant LAZARUS fait gagner beaucoup de temps, grâce à l'"Inspecteur d'Objets" et au Code Source de développement.

Votre Composant s'il est bien documenté peut devenir un atout fiable, faisant connaître votre société. Il peut aussi être sauvé de l'abandon, grâce à une communauté toujours plus avare de Composants Libres. Un Composant qui devient Libre veut soit être populaire, soit se faire connaître pour survivre. Cela n'empêche pas une activité commerciale.

Créer un Composant Libre permet de faire connaître votre Composant, mais aussi de trouver le moyen de savoir si on reste le meilleur dans son domaine, en créant une communauté sondant les utilisateurs et contributeurs.

N'espérez pas des contributeurs qu'ils collaborent sans être rémunérés. Vous serez par contre peut-être rémunéré pour des modifications importantes. Attendez de votre communauté une expertise sur votre savoir, des aides pour vous promouvoir. Ces aides ne sont peut-être pas celles que vous souhaitez. Elles permettent donc de trouver de nouveaux marchés.

TESTER SON SAVOIR-FAIRE

Un Composant est au cœur de votre savoir-faire. Si vous modifiez votre Composant cela implique forcément des réactions en chaîne dans votre Logiciel. Il est donc très intéressant de mettre en place des jeux de tests en surchargeant vos paquets de Composants.

L'EXEMPLE

FPCUnit vous permet de tester les Logiciels FREE PASCAL. Nous allons mettre en place des jeux de tests pour notre bouton de fermeture.

Dans le répertoire de votre Composants créez un dossier "tests". Dans le menu "Fichier", créez une "Nouvelle" "FPC Unit - Application de tests",en remplaçant le nom du jeu de tests "TestCase" par "TestButtons". "Enregistrez" le projet en le nommant "fpcunitbuttons", dans le dossier créé.

Ajoutez dans la clause "uses" du projet l'unité "Interfaces". Cette unité est nécessaire pour gérer les formulaires, dont nous avons besoin pour tester le bouton de fermeture du formulaire.

Dans le menu "Projet", puis "Inspecteur de Projet", ajoutez avec "+" votre paquet en "Nouvelle Condition". Ou bien ajoutez le dossier "../" dans les "Options du projet", puis "Options de compilation", puis le dossier des librairies. Cela permet de tester le Composant sans avoir à l'installer.

Le formulaire

Ajoutez un nouveau formulaire nommé "testform". Dans la partie "interface", ajoutez l'unité "u_buttons_appli".

Dans le formulaire, nous allons mettre en place le bouton que nous avons créé.

Le formulaire possède ces méthodes et variables, dont certaines existaient déjà dans l'ascendant de la fiche :

```
 { TTestForm }

 TTestForm = class(TForm)
 public
 Bouton : TMyClose;
 procedure DoClose ( var AAction : TCloseAction );
override;
 procedure RunBouton;
 destructor Destroy; override;
 end;
```

Nous n'avons pas besoin de tester le formulaire, mais le bouton. Nous créons donc le bouton manuellement, sans utiliser l'EDI :

```
procedure TTestForm.RunBouton;
begin
 Bouton := TMyClose.Create(Self);
end;
```

Il est intéressant d'utiliser d'autres moyens de création, que ceux utilisés par le développeur, afin de vérifier le Composant autrement. Ne négligez pas cependant les jeux de tests à effectuer.

Nous créons l'auto-destruction du formulaire, afin de vérifier plus tard s'il est correctement détruit :

```
procedure TTestForm.DoClose(var AAction:
TCloseAction);
begin
 inherited DoClose(AAction);
 AAction := caFree;
end;

destructor TTestForm.Destroy;
begin
 inherited Destroy;
 TestForm := nil;
end;
```

Nous testons le bouton et ne testons pas le formulaire. Nous nous assurons donc que le formulaire est correctement affecté à "nil", même si dans les sources il est censé être correctement affecté à "nil".

La classe de tests unitaires

Voici à quoi ressemble notre classe de tests :

```
 { TTestButtons }

 TTestButtons= class(TTestCase)
 protected
 procedure SetUp; override;
 procedure TearDown; override;
 published
 Procedure Tests;
 procedure TestHookUp;
```

```
 end;
```

Les deux méthodes publiées sont affectées dans les jeux des tests par l'initialisation d'unité suivante, située tout à la fin de votre unité de classe de tests :

```
initialization
 RegisterTest(TTestButtons);
end.
```

Les tests sont récupérés par le projet de tests avec cette méthode.

Voici la méthode de création des Objets testés :

```
procedure TTestButtons.SetUp;
begin
 // Initialize
 Application.CreateForm( TTestForm, TestForm );
 TestForm.RunBouton;
end;
```

Dans votre savoir-faire il est préférable de mettre les commentaires en anglais, afin d'internationaliser vos Composants.

Le formulaire a été créé. Il est préférable de le détruire par nous même :

```
procedure TTestButtons.TearDown;
begin
 // Freeing
 TestForm.Free;
end;
```

Passons aux jeux de tests :

```
procedure TTestButtons.Tests;
begin
 SetUp;
 // testing
 AssertTrue('Testing Glyph of
TMyClose',assigned(TestForm.Bouton.Glyph));
 TestForm.Bouton.Click;
 AssertTrue('TestForm of TMyClose not
```

```
visible',Assigned( TestForm ));
 TearDown;
end;
```

L'implémentation précédente est une des deux méthodes publiées qui sera appelée pour tester le bouton. Il est donc nécessaire de créer les Composants avant les jeux de tests avec "SetUp", pour les détruire à la fin avec "TearDown".

La classe TTestCase possède elle-même des méthodes permettant d'effectuer les tests en les enregistrant dans le projet.

Un jeu de test FPCUnit commence par "Assert", suivi du type de jeu de tests.

"AssertTrue" va donc tester si le deuxième paramètre est bien à "True". Quant au premier paramètre il permet de savoir quel jeu de test nous réalisons. Si vous voulez tester une valeur, alors utilisez "AssertEquals".

Nous testons bien le fait que le "Glyph" soit affecté à la création, sans avoir besoin de formulaire. Nous testons aussi la fermeture du formulaire par le bouton "TMyClose". Nous ne testons pas cependant le fait que le Glyph ne soit pas enregistré dans le formulaire.

Cette méthode publiée permet de tester si les jeux de tests sont effectivement exécutés :

```
procedure TTestButtons.TestHookUp;
begin
 Fail('Test of button finished');
end;
```

La méthode "Fail" envoie un message d'erreur. Elle devra donc être vue à l'exécution. Elle pourra être enlevée en utilisant d'autres utilitaires de tests plus évolués.

Le projet de tests

Dans le fichier "lpr" créez cette classe :

```
 { TMyTestRunner }

 TMyTestRunner = class(TTestRunner)
 // override the protected methods of TTestRunner
to customize its behavior
 public
 procedure RunTests;
 end;
```

Cette méthode permet de démarrer tous les jeux de tests :

```
{ TMyTestRunner }

procedure TMyTestRunner.RunTests;
begin
 DoTestRun(GetTestRegistry);
end;
```

Ensuite nous créons l'exécution du projet :

```
var
 Application: TMyTestRunner;

begin
 Application := TMyTestRunner.Create(nil);
 Application.Initialize;
 Application.Title := 'FPCUnit Console test runner';
 Application.RunTests;
 Application.Free;
end.
```

Comme dans un projet standard, nous créeons un Objet central, qui gère les Objets de tests cette fois ci.

Après avoir initialisé, puis nommé notre Logiciel, nous démarrons les tests. Enfin nous libérons notre Logiciel.

EXERCICE

Améliorez vos tests, grâce à la documentation de FPCUnit, ou bien grâce à FPCUnit2.

REMARQUES

Il est préférable de toucher le moins possible à certaines propriétés de paquet. Par exemple, si vous changez un chemin dans l'onglet "Chemins" d'un paquet, LAZARUS compilera mal.

Si vous ajoutez un formulaire et que vous voulez l'ajouter à votre paquet, vérifiez bien que vous ne l'avez par ajouté à un projet en cours.

CONCLUSION

Les Composants doivent être testés. Mais ils permettent de gagner beaucoup de temps grâce à votre EDI de DRA.

Les Composants permettent aussi de créer des Logiciels personnalisés et intuitifs.

À la fin vous disposez d'un moteur créant vos Logiciels grâce à une analyse uniforme, stockée dans des fichiers passifs. Les gains de temps sont minimes avec le moteur comparé au travail fourni, mais ce dernier permet de fiabiliser votre création de Logiciels.

COOPÉRATION DRA

Introduction

Lorsque son projet, ou celui d'un autre, demande l'intervention d'au moins un autre développeur, il convient d'installer un outil de collaboration. On utilise un outil collaboratif public, si nos sources sont libres, ou un outil interne ou non référencé, si nos sources sont commerciales.

Le libre et l'économie

Pouvoir améliorer puis redistribuer une œuvre permet indéniablement d'avoir exactement ce que l'on veut avec le minimum de temps. Sans le partage un travail peut être perdu.

Vous me direz qu'il faut cependant créer des emplois, et que les entreprises privatives permettent de créer des emplois. C'est une réflexion de Shadok. On a tous compris qu'il ne suffit pas de pomper pour que l'économie tourne. C'est pourtant ce que proposent beaucoup de personnes censées défendre la valeur travail.

L'économie actuelle ne fonctionne assurément pas sur la création d'emplois virtuels. L'économie actuelle fonctionne d'abord grâce à l'agriculture avec les industries améliorées par la science. C'est là qu'interviennent le domaine public et les licences libres.

Pour qu'un pays se développe il faut améliorer le fonctionnement des secteurs agricoles et industriels. Ainsi les trains, en utilisant un seul moteur pour beaucoup de personnes ou de produits, permettent de ne pas utiliser le cheval ou la voiture et son conducteur. L'association scientifique d'éléments va donc permettre d'économiser du travail pour notamment améliorer le secteur agricole.

Avec le moteur, le cheval va alors devenir un partenaire plutôt qu'un outil. Le conducteur va lui s'occuper d'autre chose, pour peu que le pays se développe, grâce aux trains par ailleurs. On va par exemple orienter le conducteur vers les métiers scientifiques ou artistiques.

Depuis la privatisation de la monnaie en 1973, on voit ainsi que la perte des lignes ferroviaires crée des bouchons dans les grandes villes tout en appauvrissant les campagnes. En effet les trains faisaient vivre la campagne grâce à cette économie de travail. L'économie de travail est donc la base de notre économie. Les trains ont été créés parce qu'on a pu capter l'énergie du charbon, plus dense que la biomasse. On a remplacé du travail humain par un travail de moteur. Avec le développement des industries et de l'association l'humain devient un esprit, pas une Ressource Humaine.

Les licences libres, qui sont l'adaptation des bénéfices du domaine public au droit d'auteur, permettent elles aussi l'économie de travail par l'association, de la même manière que le train. La licence libre permet notamment de protéger la diffusion de l'œuvre du droit d'auteur. En effet le droit d'auteur ne permet plus de protéger l'auteur. Ce dernier peut signer un contrat d'exclusivité. Cela se termine le plus souvent par une cession dans le domaine public, 70 ans après la mort de l'auteur en France. Ne parlez surtout pas d'assurance vie à un auteur censuré par l'exclusivité.

Lorsque l'économie se contracte ou lorsque le pays se développe, diffuser largement une œuvre va permettre une réutilisation rapide de l'œuvre. Vous me direz que les œuvres propriétaires peuvent être diffusées aussi très vite, grâce à du marketing et aux industries. Seulement ces œuvres se basent très souvent sur le domaine public ou certaines œuvres libres pour être vite créées. Nous naissons avec un héritage à améliorer. Créer consiste le plus souvent à améliorer. Il est très facile d'améliorer une recette plutôt qu'un plat à consommer. Ainsi les Logiciels privatifs contiennent de plus en plus de sources libres, en respect de la licence privative, dit-on.

La licence GPL, demandant à redistribuer les sources, peut d'ailleurs être un piège pour ces entreprises. C'est ainsi qu'une partie du système ANDROID est devenu libre, puisqu'il était devenu entièrement dépendant de LINUX. La concurrence va alors demander à GOOGLE de faire plus de fleurs à l'utilisateur.

La licence GPL est économiquement saine. Par ailleurs le système LINUX exige un haut niveau de sûreté, parce que quiconque peut l'améliorer. Ce haut niveau de sûreté n'était même pas respecté par GOOGLE.

N'essayez pas de créer quelque chose qui a déjà été fait. C'est ce que sont censées faire les entreprises pour respecter le non partage de la recette d'une œuvre. Les termes anglais sont Reverse Engineering ou Hacking quand on utilise une œuvre pour créer la recette. Le Hacking avec son homonyme plus récent et moins connoté sont utilisés pour percer le secret d'une œuvre, afin de pouvoir l'adapter au monde actuel pour utiliser l'œuvre. Tout ceci peut être légal ou illégal, selon le respect du droit d'auteur ou du brevet. Mais il faut aussi respecter l'économie de travail consistant à utiliser ce qui a été produit.

Une entreprise peut d'ailleurs utiliser l'économie de travail pour se défendre face à une entreprise privative. En effet, les licences libres permettent de se protéger du privatif et de son extension l'obsolescence programmée. Sans la recette de l'œuvre impossible de réparer correctement certaines parties de l'œuvre. On peut se tromper et détruire l'œuvre. Il n'y a plus d'économie de travail. Le pays ne se développe pas. L'entreprise corrompue doit alors permettre la réparation, pour peu que la production soit faite dans le pays de vente. Produire et vendre dans le même pays permet de respecter le producteur et donc le consommateur.

GNU LINUX REPLICANT, clone entièrement libre de LINUX ANDROID, permet ainsi de sauver la diffusion d'un matériel privatif, en lui ajoutant d'ailleurs plus de sûreté et de pérennité. Beaucoup d'autres systèmes libres permettent de faire cela. Les libristes vont d'ailleurs adorer installer un LINUX sur un matériel privatif. En faisant cela ils prolongent la durée de vie du matériel et permettent donc l'économie de travail. Ils se protègent et protègent les autres du privatif.

Le domaine public et les licences libres permettent l'économie de travail, notion essentielle de notre économie, grâce à la diffusion rapide, la sûreté du partage, la pérennité. Nous sommes des individus sociaux et créatifs capables de lier de plus en plus vite et et plus en plus d'éléments, grâce à l'association d'idées et de personnes, avec la réflexion scientifique selon Platon.

POURQUOI PARTICIPER ?

LAZARUS est l'outil DRA libre le plus complet. Il a sauvé DELPHI. Ainsi de nombreux projets de Logiciels industriels français peuvent être sauvés. En effet le byte-code que l'on crée avec le langage JAVA est plus léger en Pascal, alors que beaucoup d'entreprises françaises dont AIRBUS sont anciennement passées de PASCAL à JAVA, pour revenir sur PASCAL. JAVA est très utilisé en ce moment, alors qu'une application PASCAL est compatible JAVA et plus rapide.

La communauté PASCAL est active parce qu'il est facile d'apprendre le PASCAL Objet. On crée vite des interfaces homme-machine. Il devient ensuite ingénieux de bien connaître les Composants Objets. La faille de beaucoup d'entreprises a été de n'être que des consommateurs d'EDI de DRA, alors qu'il existait des projets libres, surtout en PASCAL.

En 2006, on me disait souvent de ne surtout pas participer aux projets libres, en commençant par la simple utilisation. C'est normal qu'un projet libre fasse moins de choses qu'un projet payant. Cependant s'il fait le nécessaire c'est lui qu'il faut choisir. On m'avait indiqué qu'il y avait beaucoup plus de gadgets dans le projet DEV EXPRESS et que c'était une raison pour le prendre. Cependant on a pu améliorer une grille partagée pour avoir exactement ce qu'on voulait. Impossible de participer à DEV EXPRESS aussi facilement.

Sans la diffusion partagée, le droit d’auteur ne permet pas de protéger un projet esseulé. Le fait que DELPHI partage ses sources a permis un début de pérennité de l’outil DELPHI. Ce qui a sauvé DELPHI c’est le partage d'INTERBASE qui a créé FIREBIRD. FIREBIRD est aujourd’hui lié a LIBRE OFFICE. Les projets Libres dont JEDI ont incité à créer LAZARUS par un compilateur PASCAL Libre. Les projets INDY et INTRAWEB ont permis de pérenniser DELPHI par le partage.

Depuis DELPHI 2006 jusqu'à DELPHI XE, le projet DELPHI capotait, comme pour tous les outils RAD qui étaient à l'époque propriétaires. On voit selon la

déchéance NOKIA que MICROSOFT détruit les autres projets pour son hégémonie. Beaucoup de grosses entreprises agissent comme cela à cause de la théorie des jeux. La théorie des jeux ne permet pas d'anticiper sur une entreprise voyant sur le long terme.

Le tout libre n'est possible que dans des pays qui se développent. On peut prendre les exemples récents de l'Asie. Dans ces pays beaucoup de travaux ont été dans le domaine public ou sont libres. Là-bas, il n'y avait pas besoin de penser à trouver de l'argent mais à produire des richesses, utiles donc. Les producteurs de ces richesses utiles étaient alors suffisamment rémunérés. Dans ce cas les entreprises voient sur le long terme. Ça n'est pas le cas chez nous depuis l'année 1973, lorsque la monnaie a été privatisée.

C'est bien le projet libre LAZARUS qui a sauvé le projet DELPHI. En effet, la version EMBARCADERO DELPHI XE3 pour iOS s'est inspirée de LAZARUS. DELPHI XE5 a pu s'inspirer de LAZARUS pour ANDROID, notamment du compilateur de byte-code.

Pendant un certain temps, LAZARUS permettait de passer au 64 bits alors que DELPHI ne le permettait pas. Une entreprise qui m'avait téléphoné m'avait dit qu'elle attendait la prochaine version DELPHI pour passer au 64 bits. Or il suffisait de participer activement ou financièrement au projet LAZARUS pour avoir non seulement le 64 bits, mais la puissance de GNU LINUX, voire aujourd'hui la comptabilité de la plate-forme sur laquelle vous travaillez.

Actuellement EMBARCADERO accumule les licences achetées en France. Beaucoup d'entreprises françaises ont acheté DELPHI XE3 et DELPHI XE5. Pourtant LAZARUS aurait avancé à pas de géants avec ce genre d'aide financière. En plus l'argent aurait été pérennisé. DELPHI a besoin de LAZARUS tout comme LAZARUS a eu besoin des Composants Libres créés sur DELPHI.

PARTICIPER

Une fois que l'on a créé ses Composants et qu'ils évoluent correctement, on peut enfin participer aux projets LAZARUS.

Tout projet est Open Source afin que d'autres y participent. Seulement pour participer au projet LAZARUS il faut déjà améliorer des Composants. Vous avez vu, dans le chapitre permettant la création de son savoir-faire :
Qu'il faut avoir en tête la structure des Composants ancêtres.
Que toute nouvelle propriété doit avoir un nom bien choisi.
Qu'il faut connaître l'anglais.
Qu'il faut documenter et commenter.
Qu'à terme on pense en Objet, avec des outils adéquates.
Qu'à terme on sait s'adapter à la machine, avec plein d'astuces.

Nous vous avons donné des clés pour optimiser votre Code. À vous de trouver de nouvelles astuces.

Plutôt que d'ajouter sans avoir d'idée sur la façon de le faire, il est préférable de faire avancer le projet LAZARUS, et de contacter des développeurs. Tout développeur utilisant LAZARUS peut vous aider. Les passionnés documentent et essaient d'être disponibles. Seule l'objectivité permet de s'améliorer pour participer à un projet.

On commence avec un Composant à déboguer, puis on ajoute une fonctionnalité, puis on optimise son Composant sans cesse, puis on partage son Composant pour être critiqué.

PROTÉGER SES DROITS

Afin de diffuser un Logiciel il faut protéger ce qui est important en :
S'envoyant un recommandé électronique décrivant son Composant avec les sources. Le recommandé avec signature dure 10 ans.
Envoyant son nouveau concept que l'on compte créer dans son Logiciel en enveloppe SOLEAU à l'INPI. Cela ne dure que 5 ans mais est renouvelable.
En publiant un livre blanc en Dépôt Légal.

PRINCIPE

Les sources sont accessibles à tous les développeurs participant au projet, grâce à un serveur de sources accessible.

Les développeurs, à chaque fois qu'ils veulent modifier une unité, mettent à jour leurs unités locales, sur leur ordinateur. Puis ils modifient.

Une fois l'unité modifiée, et après avoir testé, il est préférable de transférer, vers le serveur central, toutes les sources modifiées. En effet, les sources sont souvent imbriquées.
Les autres développeurs accèdent aux nouvelles sources.

POUR UNE PETITE ÉQUIPE

Lorsque l'équipe qui travaille sur les sources est petite, il est intéressant d'installer un outil de collaboration avec jeton exclusif.

Le fonctionnement en jeton exclusif est simple : Si quelqu'un veut modifier une unité, il a l'exclusivité sur la modification.
Ainsi il n'y aura pas à fusionner la même unité modifiée à deux endroits différents.

Par exemple JEDI VCS est un outil collaboratif libre permettant la distribution de sources par jetons exclusifs.

POUR UNE GRANDE ÉQUIPE

Lorsque beaucoup de développeurs participent à un projet, il convient de choisir un outil collaboratif permettant la comparaison et la fusion de sources.

Le principe est simple : Quiconque peut modifier quelque source que ce soit.

Si chacun ne modifie pas en même temps les mêmes sources, l'outil s'utilise simplement.

Il arrive que certains modifient en même temps une source. Alors le premier qui publie ses unités n'a rien à faire. Celui qui est en retard doit fusionner ses sources avec celui qui a publié avant, puis tester évidemment.

Il est préférable d'utiliser ce genre d'outils avec des alertes mails, afin que les développeurs soient au courant de ce qui est modifié.

Un patch est un ajout de Code ou de Sources, s'insérant automatiquement dans le Code ou les Sources existantes, en fonction d'une ou de plusieurs versions de départ.

En 2015, deux très bon outils collaboratifs s'appellent MERCURIAL et GIT. MERCURIAL utilise TORTOISE HG comme client. Il permet de réaliser des patches facilement, tout en distribuant les sources à beaucoup de développeurs.

Les outils SVN et CVS n'ont pas autant de répondant, ni autant de possibilités que MERCURIAL et GIT. MERCURIAL s'utilise plus facilement que GIT, notamment par le fait qu'on n'a pas forcément besoin de serveur de sources.

CHOISIR SON OUTIL

Un bon outil de collaboration permet de :
Créer des archives permettant de recréer les anciennes versions
Communiquer facilement entre les développeurs
Créer des patches facilement
Créer plusieurs versions de son exécutable
Travailler sur les différents systèmes d'exploitation
Travailler simplement, sans avoir à se soucier des difficultés d'un partage

Les outils TORTOISE sont intégrés à votre bureau.

TORTOISE SVN et TORTOISE CVS impliquent une compatibilité de l'outil de programmation.

TORTOISE HG pour MERCURIAL s'utilise facilement.

Vous avez ci-après un exemple d'intégration d'un outil TORTOISE dans le bureau :

Les outils TORTOISE sont intégrés au bureau graphique

RAPID SVN sous LINUX et WINDOWS, avec WIN CVS sous WINDOWS, permettent de répondre au multi-plates-formes de LAZARUS. Ces outils graphiques sont simples d'utilisation.

LES FORGES

Une forge est un serveur de sources regroupant les sources de différents savoir-faire, de différents Logiciels.

Une forge partagée regroupe des sources mondiales et n'accepte que des sources partageables. Les licences des Logiciels de ces forges sont des licences Open Source.

Serveurs gratuits

Il existe différentes forges à sources partagées.

www.sourceforge.net regroupe des sources partagées. Il peut utiliser SVN, CVS, MERCURIAL et GIT.

www.bitbucket.org regroupe des sources partagées. Il peut utiliser SVN, CVS, MERCURIAL et GIT.

www.github.com regroupe des sources utilisant de préférence GIT.
Il peut utiliser SVN, CVS, MERCURIAL.

Serveur français

En France www.adullact.net est une forge partagée, réservée aux collectivités et administrations.

En cherchant le projet libre de forge fusionforge, on trouve d'autres forges partagées.

L'INRIA a mis en place un projet de regroupement de sources libres.

ASTUCES POUR LA CONCEPTION PARTAGÉE

Tous les informaticiens RAD savent qu'il est important d'utiliser des existants. Seulement les existants partagés sont gratuits et donc leur partage non rémunéré. Une société idéale partagerait des savoirs-faire grâce à l'état-nation par des chercheurs. Or l'INRIA a été privatisé en 2015 et ne remplit plus cet objectif d'économie de travail, base de l'économie humaine.

Pour concevoir ou améliorer un savoir-faire de composants, il faut avoir en tête la structure ainsi que les chemins engagés et envisagés par le savoir-faire, qui doivent être écrits respectivement par un automate et l'auteur du savoir-faire.

Si vous voulez demander quelque chose à quelqu'un il y a le donnant-donnant : On aide ou on propose d'améliorer le savoir-faire. Créer une documentation est accessible à un débutant. C'est d'ailleurs préférable qu'un débutant crée la

documentation. En effet, c'est quand on apprend qu'on est le plus accessible aux débutants. Si le wiki n'est pas disponible sur le gestionnaire de versions, sachez qu'il existe des wikis pour chaque outil partagé qu'utilise le savoir-faire.

Quand un savoir-faire est partagé, l'auteur adore avoir des retours d'expérience lui permettant d'améliorer son savoir-faire. Comme tout ce qui est important est visible par quiconque en même temps grâce à la technique informatique, donner votre avis ou mieux, proposer une issue à un savoir-faire, est attendu, même si vous êtes débutant.

Sachez que tous les informaticiens vont vers l'économie de travail, base de notre économie. Donc ne vous attendez pas à un travail peaufiné de la part d'un seul informaticien. Il faut obligatoirement être deux au moins pour réaliser une économie de travail intéressante. En effet, un informaticien qui sait compiler seul peut aider facilement à compiler un informaticien qui a compilé seul son savoir-faire. Lisez à ce sujet Henry Charles Carey et Robinson Crusoé.

Pour aider un auteur de composants partagés à compiler, il suffit de s'abonner gratuitement à son gestionnaire de versions. Alors on compile entièrement, puis on efface les unités compilées et on recompile. On ajoute une issue au gestionnaire de versions s'il y a des problèmes. L'auteur peut alors nous proposer de participer à son projet en fonction de vos compétences. Renseignez-vous alors sur ses autres compétences, qui peuvent peut-être vous aider.

Un auteur de composants s'intéresse à la technique. Le challenge technique est important et peut même faire évoluer un savoir-faire, afin que celui-ci gagne des composants. Dans ce cas, il est possible de retrouver des composants qui ne fonctionnent pas ou mal, parce que le spectre de compilation de l'outil est conséquent.

Si vous participez à un savoir-faire partagé vous pouvez ainsi aider des milliers de personnes qui distribuerons leurs logiciels à des millions de personnes. Le logiciel partagé c'est l'association, association plus rapide que l'association par l'argent, car proche de la vitesse de la lumière.

Pour améliorer des composants il s'agit de comprendre l'objet. Avoir lu un cours avancé de programmation de Composants Objets peut vous permettre de vous

faire passer pour le meilleur informaticien qui soit, face à l'auteur du partage.

Si vous ajoutez des composants à ce savoir-faire ce sera à vous de vous occuper de ces Composants. Vous pourrez alors peut-être plus tard reprendre en main l'ensemble du savoir-faire. Vous deviendrez l'auteur principal du savoir-faire, créant alors vos propres objectifs, pour peu qu'ils vous servent.

AMÉLIORER LAZARUS

LAZARUS est bien conçu. Les développeurs de LAZARUS vérifient la rapidité, la portabilité, la continuité de leur outil. Les modifications qui ne s'assemblent pas de la même manière que DELPHI sont des améliorations structurelles. Elles permettent par exemple d'améliorer la lecture d'images, de réutiliser des API graphiques. LAZARUS c'est une réutilisation du meilleur des librairies existantes.

Seulement si on s'arrête là LAZARUS ne peut pas s'améliorer. Les Composants de DRA sont non seulement intuitifs, mais permettent d'améliorer le Polymorphisme.

Utilisons l'objectivité. Vous voyez que les Composants LAZARUS fournis ne suffisent pas pour créer un Logiciel. Il est possible de participer à LAZARUS en traduisant un Composant Libre DELPHI vers LAZARUS. Le Composant peut rester compatible DELPHI grâce aux directives de compilation. Ainsi dès que la source diffère on ajoute une directive FREE PASCAL.

Si votre Composant est primordial pour une utilisation standard de LAZARUS, vous pouvez proposer votre Composant en Patch pour LAZARUS. Les développeurs de LAZARUS vérifient ensuite la licence, et le respect de la programmation par Objets. Ensuite si le Composant est nécessaire au projet il est validé.

Sinon vous pouvez diffuser votre Composant sur un site Web regroupant des Composants. Faites-le connaître par le réseau LAZARUS. Vous pouvez diffuser votre Composant grâce à onlinepackagesmanager et www.lazarus-ide.org. Il est

nécessaire de créer un message en anglais, sur ce site web, dans le forum contenant les messages de "Tierce partie".

Si votre Composant est en cours d'importation il est possible de devenir développeur LAZARUS. Vous devenez alors un auteur de LAZARUS.

Il est aussi possible de faire évoluer FREE PASCAL pour qu'il utilise plus de machines. Vous intégrez alors un compilateur Libre à LAZARUS. Ensuite l'équipe LAZARUS va ajouter cette nouvelle architecture à votre IDE Libre.

CRÉER UNE FONDATION

Il est constaté que ce sont les développeurs indépendants qui partagent le plus. Pourtant, au moment où on économise sur les budgets, la solidarité pourrait exister entre les PME.

Il fut une époque où le Produit Intérieur Brut n'incluait pas les services. C'est l'inclusion des services dans le PIB qui a créé la bulle des Startups du web. Pourtant les Startups du web sont dépendantes des richesses réellement produites.

Les services devraient pourtant permettre de mieux produire. Notre économie fonctionne vers l'utile comme cela. La plus grande richesse ce sont les industries. On doit les protéger. Les industries créent des emplois qualifiés. Elles nécessitent et créent l'éducation qui développe alors l'élève. On voit en ce moment des services qui se créent avec des emplois peu qualifiés. Ces services embauchent souvent des sur-diplômés.

Actuellement des lobbies se mettent en placent autour des métiers nécessitant d'individualiser les citoyens pour se développer. Ces métiers font en général du trafic. En effet, selon l'économie physique, le commerce consiste à vendre directement, en passant par le minimum d'intermédiaires. Cela enrichit l'individu.

Le trafic consiste à augmenter les prix en faisant pression sur la production, d'après l'économiste Henry Charles Carey. Cela contracte l'économie. On se sert alors de l'humain comme d'une ressource. Les métiers qui individualisent l'individu prennent alors de plus en plus d'importance. Cela finit par l'implosion de l'économie, car ces métiers font pression sur la production.

Les citoyens et donc les entreprises se doivent d'empêcher ces trafics en défendant le bien commun. Cela est la fois bon pour le développement et donc pour l'éthique. On ne cherche pas le gain à court terme, mais la capacité a élaborer la société, grâce au bien commun que sont par exemple le domaine public et les licences Libres.

Dans le bien commun, il y a aussi le savoir et ce qui sert à l'économie de travail, comme les services et industries publics. Les industries et services publics intégreront les métiers faisant pression sur le développement d'un pays. On cherche ainsi à utiliser son intelligence pour mieux produire de la richesse tangible. Retrouver le sens du bien commun permet à la fois de développer un pays afin de trouver le bonheur pour devenir ingénieux, en cherchant sur nos limites.

On peut ainsi partager un savoir-faire pour créer une fondation. Il est alors nécessaire que des entreprises se mettent d'accord pour faire évoluer le savoir-faire. Une inter-dépendance entre des entreprises concurrentes peut alors se créer. Cela permet, en période de contraction de l'économie ou en plein développement, d'économiser sur le travail afin de produire de la richesse réelle.

La solidarité existe réellement quand un pays se développe. Dans ce cadre on ne parle plus de licences libres, mais réellement du domaine public. Le domaine public n'existe réellement que dans les pays qui se développent, comme en Russie ou dans les pays asiatiques en 2015. En effet quand le commerce existe réellement, la population s'enrichit réellement. Il n'y a pas besoin de protéger des acquis. On est dans la profusion, la diversité et surtout l'association. La science fait alors de grands bonds en avant. La créativité est favorisée.

C'est l'association et l'économie de travail qui permettent le développement. Notre économie consiste à associer des éléments entre eux. Cela nécessite que

nous associons entre nous et en nous. Le domaine public et les licences libres permettent d'accroître l'association et l'économie de travail avec le DRA.

Source : Les livrels et livres sur l'économie et la créativité de www.liberlog.fr.

GNU LINUX et FIREFOX démontrent la force de l'association.

DE PASCAL À FREE PASCAL

CREATIVE COMMON BY SA

INTRODUCTION

FREE PASCAL permet de créer des savoir-faire multi-plates-formes à partir de votre savoir-faire WINDOWS comme DELPHI ou TURBO PASCAL.

À partir de ces savoir-faire ou Logiciels, on crée des librairies FREE PASCAL. Cependant cela demande l'adaptation d'une partie des Sources.

MIGRATIONS

Beaucoup d'éditeurs utilisent de vieilles versions de leur outil. En effet ils ne savent pas comment migrer vers une nouvelle version du vieil outil. Aussi, en période de contraction de l'économie, les entreprises ont du mal à voir sur le long terme.

Il est pratiquement impossible de migrer un Logiciel non RAD vers de nouveaux Frameworks. Avec un outil non RAD, le choix du Framework est donc primordial. Ce Framework doit être suffisamment simple et ingénieux pour gagner du temps.

W4 EXPRESS est un moteur qui transforme une analyse en Logiciel avec un moteur Logiciel. La personnalisation peut se programmer avec des plugins appelés "behaviors". Il permet aussi d'ajouter des systèmes d'exploitation en ayant le minimum à programmer. Ainsi les intégrateurs W4 EXPRESS ont pu bénéficier de ANDROID et IOS en 2013, sans rien avoir à programmer. Il existe dorénavant en 2018 beaucoup de moteurs BPM libres en JAVA.

Il existe ATK FRAMEWORK ou JELIX JFORMS en PHP. Le Framework ATK permet aussi de créer un Logiciel léger à partir des données. Il est possible

d'utiliser les fichiers passifs des moteurs libres de BPM pour créer les interfaces JELIX ou ATK. Ces Frameworks sont faits pour que vous y participiez. Ils permettent de créer des Logiciels fiables vite.

Malgré tout, le problème des langages non RAD, c'est qu'ils ont tendance à complexifier la programmation. Ainsi la difficulté à programmer va créer de l'incompréhension chez les programmeurs, conduisant à des Sources mal élaborées, voire mal présentées.

Avec un outil de DRA il est possible de migrer plus facilement, si son outil de DRA possède beaucoup de Composants avec des Sources. En effet les Sources permettent non seulement de fiabiliser son Logiciel, pour trouver les défaillances, mais aussi des Sources pourront facilement migrer vers une nouvelle version de l'outil.

Par contre surcharger les Composants sera nécessaire. En effet, créer des composants permet de commencer la migration en créant à la fois un savoir-faire Logiciel, mais aussi en permettant un transfert vers le nouvel outil petit à petit. Le savoir-faire créé permettra par ailleurs de trouver de nouveaux clients. On crée alors une méthode et des mots-clés de ce que l'on avait élaboré. Cela permet de créer différentes versions de ses Logiciels plus facilement, de façon centralisée, simple, structurée et intuitive avec son outil de DRA.

Si des Composants n'existent plus sur la nouvelle version de l'outil et que les Sources ne sont pas Libres, il sera nécessaire aussi de surcharger d'autres Composants. On crée les propriétés manquantes avec la surcharge. On crée un Logiciel permettant de remplacer certains mots clés par d'autres. C'est l'outil de migration.

Le savoir-faire permettra aussi d'envisager une automatisation utilisant l'ingénierie pilotée par les Modèles. Contrairement à l'Architecture Pilotée par les modèles, on ne crée pas de Sources risquant de ne plus recréer le modèle. L'Ingénierie Pilotée par les Modèles va permettre de créer un moteur lisant l'analyse pour créer le Logiciel, en automatisant les Composants par la surcharge.

On surcharge aussi les Composants pour permettre plus de possibilités au

Logiciel. Par exemple en automatisant la création de rapports, il est maintenant possible, dans le savoir-faire libre sur LAZARUS nommé EXTENDED, de permettre à l'utilisateur d'adapter la largeur des colonnes de son impression. EXTENDED permettra de migrer un Logiciel fait en DELPHI vers LAZARUS.

LAZARUS permet de passer à GNU LINUX. LINUX vous apporte la rapidité du noyau LINUX, démontrée par la plate-forme STEAM. Vous disposez aussi de la rapidité des dernières partitions adaptées aux disques durs récents. Il y a aussi la sécurité UNIX et celle des Logiciels partagés. Par contre il sera intéressant de rester compatible DELPHI. En effet DELPHI est plus rapide que LAZARUS sur WINDOWS, même s'il utilise plus de ressources mémoire.

LAZARUS permettra de migrer régulièrement, donc plus facilement. En effet il n'y aura plus besoin d'acheter un ensemble de licences pour le développeur. Par contre les langages Libres demandent souvent une participation partagée lors des migrations, surtout lorsqu'on passe de DELPHI à LAZARUS.

De TURBO PASCAL vers FREE PASCAL

Pour transférer un Logiciel TURBO PASCAL textuel il suffit de réutiliser les bonnes unités. Elles se trouvent nécessairement dans LAZARUS ou FREE PASCAL.

Pour transférer un savoir-faire TURBO PASCAL graphique :
Insérez la partie graphique dans une fenêtre et adaptez la taille graphique à la fenêtre
Ou bien recréez la partie graphique avec les Composants visuels

Mes bases ne peuvent pas être migrées

Il y a différentes raisons pour qu'une source de données ne soit pas sur la dernière version du système d'exploitation :
Il y a mieux comme source de données.

La source de donnée n'était pas libre et elle a été censurée.
Vous n'avez pas cherché au bon endroit.

De vieux Composants Libres peuvent suffire. Vous en avez de répertoriés sur le site www.lazarus-components.org. Cependant LAZARUS permet d'utiliser des Systèmes de Gestion de Bases de Données. Si votre base à migrer ne peut pas être utilisée avec les Composants visuels c'est que votre Base de Données ne peut facilement garder l'intégrité des données. Il est même possible qu'elle ne puisse pas accepter un nombre suffisant d'utilisateurs. La migration vers LAZARUS demandera alors d'utiliser les Composants visuels avec les accès aux données hérités de TDataset. Les TDataset permettent d'utiliser les SGBD définis plus haut.

En effet permettre une intégrité des données avec les clés et une utilisation centralisée est possible avec les SGBD. FIREBIRD permet en plus de garder l'ancienne utilisation sur un seul ordinateur.

GESTIONNAIRE DE DONNÉES

Si vous souhaitez beaucoup d'utilisateurs ou de données créez un lien vers un Système de Gestion de Base de Données, comme FIREBIRD, SQLITE, MARIA DB, voire ORACLE.

FIREBIRD et SQLITE sont des S.G.B.D. mi-lourds suffisants pour tout Logiciel installé facilement. Ils s'utilisent en Embarqué.

MARIA DB permet lui de faire évoluer votre serveur de données en serveurs de données répartis nommés "clusters". ORACLE lui permet de créer un seul serveur de données très lourd.

Il est possible d'utiliser des gestionnaires de données mi-lourds, si votre entreprise est répartie. On crée un réseau maillé permettant de dupliquer les informations. Le siège prend le pas sur les filiales.

Spécialiser vos Composants

Il est intéressant d'utiliser des Composants spécifiques à une base de données car leur création a été optimisée pour une seule source de données.

Si vous changez de Composants d'accès aux données il est important de savoir si votre Logiciel pourra fonctionner de la même manière. Par exemple l'accès aux données BDE permet de modifier une union de tables pour sauver dans la table maître.

Les Composants spécialisés IBX pour FIREBIRD demandent à créer automatiquement des requêtes afin de personnaliser la modification, la suppression, pour créer des ensembles complets de données.

Des projets comme MAN FRAMES permettent de changer et d'utiliser facilement vos Composants d'accès aux données. MAN FRAMES permet de gérer plusieurs paquets d'accès aux données. Le fichier INI configurera alors votre accès aux données automatiquement.

DE DELPHI VERS LAZARUS

Passer de DELPHI à LAZARUS consiste essentiellement à réutiliser des Composants LAZARUS à la place de Composants DELPHI. Il s'agit donc de migrer vers des Composants Libres. EXTENDED pour LAZARUS fut compatible avec DELPHI XE 5.

Il est possible de remplacer les Composants non traduits par de nouveaux Composants LAZARUS. Si vous voulez et si vous disposez des Sources, vous pouvez aussi traduire des Composants DELPHI vers LAZARUS.

Vous aurez sans doute à ajouter les unités communes LAZARUS comme "LCLType" ou "LCLIntf". Ces librairies remplacent certaines unités de DELPHI comme l'unité "WINDOWS".

Pourquoi n'a-t-on pas gardé certaines unités DELPHI ?

Les librairies LAZARUS possèdent une nomenclature. Une librairie possédant le mot "win" est une librairie système spécifique à la plate-forme WINDOWS. Elle ne doit pas être utilisée directement dans vos programmes. Les unités "gtk" ou "qt" servent à GNU LINUX, "carbon" et "darwin" à MAC OS, "unix" à UNIX.

La configuration de construction de LAZARUS vous montre l'ensemble des plates-formes indiquant les unités à éviter. Dans les Sources de LAZARUS, vous voyez des unités commençant pas ces plates-formes ou comportant ces noms de systèmes.

Il est possible que son Logiciel reste compatible DELPHI, grâce aux directives de compilation. Cela permet d'alléger l'exécutable WINDOWS.

Cependant, si vous utilisiez avant des tables pouvant difficilement être utilisées en réseau, il est préférable de migrer vos tables DBASE ou PARADOX vers FIREBIRD. Cela sera beaucoup plus facile à gérer ensuite. En effet FIREBIRD peut aussi bien s'utiliser sans serveur qu'avec, ceci avec des composants plus faciles à gérer parce que complets.

Il existe des freewares pour passer à FIREBIRD. Préférez cependant les Logiciels libres comme FLAMEROBIN. En effet ces Logiciels peuvent être modifiés plus facilement. Cela ne veut pas dire qu'ils évoluent plus vite. Par contre ces outils vont vers l'utile.

Un convertisseur LAZARUS est disponible complètement sur www.lazarus-components.org. Elle permet de remplacer des Composants comme DEV EXPRESS afin de convertir un projet DELPHI. Vous avez dans les "Outils" un convertisseur simplifié de projet, permettant de finaliser ensuite.

Une fois que l'on a changé les Composants il existe des outils LAZARUS permettant de continuer la traduction. Il sont dans le menu "Outils" de LAZARUS. Il vous reste alors à finir à la main.

Les directives de compilation

Les instructions de compilation permettent de porter son programme vers les différentes plates-formes. Elles définissent une façon de compiler l'unité. Elles sont donc à placer au début de l'unité.

Les directives de compilation doivent être idéalement placées dans un fichier unique portant l'extension ".inc".

Les directives de compilation permettent d'alléger efficacement les sources. Aussi l'optimisation des sources permet de gagner du temps.
Ainsi, vous pouvez facilement éluder un paquet en le déliant. Vous éludez ainsi chaque paquet en créant la directive associée. Comme cela vos paquets pourront facilement être allégés, ce que ne peut pas faire JAVA ou PHP.

Exemple

Voici une version allégée du fichier ".inc" des Composants "EXTENDED".

```
{.$DEFINE CSV}
{.$DEFINE RX}
{.$DEFINE MAGICK}
{.$DEFINE IBX}
{$IFDEF FPC}
 {$DEFINE ZEOS}
{$ELSE}
 {$DEFINE ZEOS}
 {$DEFINE JEDI}
{$ENDIF}
```

Ce fichier ".inc" permet de définir quels Composants vont être éludés dans DELPHI et FREE PASCAL COMPILER sans avoir à supprimer du code.

Une directive de compilation peut devenir un commentaire, si le $ ne suit pas

l'accolade.

Lorsqu'on est sur les deux plate-formes on utilise ici les Composants d'accès universel aux données ZEOS, présents sur les deux plates-formes.

Lorsqu'on est sur LAZARUS on peut utiliser ZEOS. ZEOS permet d'accéder à beaucoup de bases sur LAZARUS. Les Composants ZEOS non spécialisés ne sont pas les plus rapides malgré tout.

Lorsqu'on est sur DELPHI on utilise JEDI qui n'est pas encore complètement traduit sur LAZARUS.

Des unités du projet "EXTENDED" comporteront l'inclusion de ces directives :

```
{$I ..\extends.inc}
```

Le fichier des directives doit être placé à la racine de votre projet afin de centraliser. On descend alors d'un répertoire comme ci-dessus.

Vous pouvez utiliser aussi les directives :
IFNDEF pour vérifier la non-existence d'une définition.
UNDEF pour supprimer temporairement une définition.

Ces directives servent notamment à définir des OU à partir d'ensembles de définitions.

Compatibilité DELPHI

Voici une instruction de compilation FREE PASCAL :

```
{$mode DELPHI} // Compilation compatible
DELPHI
```

Il existe des DELPHI gratuits, permettant de réaliser des applications non commerciales. Si quelqu'un vous demande d'être compatible DELPHI vous pouvez le faire grâce à une vieille licence de DELPHI 2005. L'éditeur de

DELPHI reconnaît les fichiers UTF8 à partir de 2005.

L'instruction de compilation, qui permet à FREE PASCAL de rendre le Code Source compatible DELPHI, n'est pas lisible par DELPHI.
On peut y ajouter une directive de compilation pour DELPHI si on l'utilise.

Nous avons vu la notion de pointeur dans le chapitre sur la **Programmation Procédurale avancée**.
Le mode DELPHI permet de supprimer une partie de la notion d'adresse de pointeur. Cela n'est pas gênant, au contraire, car DELPHI et LAZARUS protègent les pointeurs.

Les pointeurs sont des adresses en mémoire, redirigeant vers un autre endroit de la mémoire. Ils permettent de manipuler les Objets. En mettant en mode DELPHI vous pouvez oublier les pointeurs en partie, car le compilateur FREE PASCAL n'est pas entièrement compatible avec ce mode.

Le développeur a besoin de manipuler des Objets, pas de gérer des pointeurs. Le pointeur peut cependant être utile pour scruter les tableaux. PASCAL Objet permet toujours cela.

Il est possible en PASCAL d'éluder totalement les affectations de pointeurs, grâce à cette directive à ajouter à tout formulaire :

```
{$IFDEF FPC}
{$mode DELPHI}
{$R *.lfm}
{$ELSE}
{$R *.dfm}
{$ENDIF}
```

Ces directives peuvent être mises en tout début d'unité et une seule fois. Elles définissent comment va être compilée l'unité.

Un fichier "dfm" est une correspondance DELPHI du fichier "lfm" avec LAZARUS. Les fichiers "dfm" et "lfm" contiennent les informations des Composants chargés dans un module de donnée, ou un formulaire. Les Composants de ces entités sont visibles dans l'"Inspecteur d'Objets".

Ici l'instruction de compilation "Mode DELPHI" est exécutée si on utilise le Compilateur FREE PASCAL. Aussi on charge le fichier "lfm". Sinon on charge les Composants contenus dans le fichier "dfm". Si votre unité n'est ni un formulaire, ni un module de données, vous pouvez enlever les directives de chargements de fichiers "lfm" et "dfm".

Cette directive de compilation permet de placer une instruction DELPHI dans le Code :

```
{$IFNDEF FPC}
const DirectorySeparator = '\';
{$ENDIF}
```

Ci-avant on définit le caractère de séparation de répertoire quand on est sur DELPHI. Cette déclaration existe déjà en FREE PASCAL. Seulement certaines versions de DELPHI, compatibles uniquement avec WINDOWS, ne déclarent pas de constante de séparation de répertoire.

En effet lorsqu'on crée un Logiciel multi-plates-formes, il est nécessaire de penser aux différences entre les différentes plates-formes. Aussi penser aux erreurs pouvant se produire permet de sécuriser son Logiciel.

TRADUCTION DE COMPOSANTS

Pour traduire un Composant DELPHI vers LAZARUS, il est préférable que ce Composant reste compatible DELPHI, grâce aux directives de compilation. Cela permet de participer au projet existant en centralisant les Sources.

Remplacez le Code graphique et le Code système, avec les liens vers les librairies WINDOWS, par du Code Source LAZARUS. LAZARUS possède une large bibliothèque de Codes Sources. Des Composants Libres DELPHI peuvent être traduits vers LAZARUS.

Le Code graphique LAZARUS utilise les différentes librairies Libres de chaque plate-forme. Ces différentes librairies sont homogénéisées en des librairies

génériques.

Les unités à ne pas utiliser sont les unités spécifiques à une seule plate-forme, comme les unités :
"win" pour WINDOWS
"gtk" ou "qt" pour GNU LINUX
"carbon" ou "cocoa" pour MAC OS
"unix" pour UNIX

Si vous êtes obligé d'utiliser une de ces unités, sachez que vous aurez peut-être une unité générique dans la prochaine version de LAZARUS. Votre Composant sera compatible avec la seule plate-forme de l'unité non générique utilisée.
Si vous ne trouvez pas ce qu'il vous faut, recherchez dans les Sources LAZARUS ou FREE PASCAL. Contactez sinon un développeur LAZARUS. Il vous indique alors ce que vous pouvez faire.

RETROUVER UN MOT

Abstraction	L'Abstraction permet le Polymorphisme en orienté Objet en créant une classe abstraite qui sera renseignée avec ses classes filles.
Application	Logiciel permettant de réaliser une ou plusieurs tâches.
Bogue ou Bug	Anciennement, ce terme anglais désigne la punaise, qui mange les circuits électroniques. Cette punaise créait des erreurs. Les programmeurs se sont déchargés alors des erreurs dans le Code, pour les attribuer à cet animal.
Byte Code	Les fichiers ".class" JAVA contiennent du Byte Code. Le Byte Code, comme avec Python, c'est du Code semi-compilé. Compiler du Byte Code permet de résoudre les changements de génération de processeurs dits RISC, les processeurs ARM notamment (cf **Cross-Compilation LAZARUS**).
Client/Serveur	Interface logicielle connectée à un serveur de données au sein d'un réseau interne. On peut aussi appeler ce genre d'interface du deux tiers. Un serveur de données dialogue avec son application.
Code	Ensemble de 0 et de 1 créés par le compilateur. Le Code sert à agir sur l'ordinateur.
Compilateur	Programme reprenant les différentes Sources de Code pour les compiler en un langage machine, des 0 et des 1 exécutables par le processeur.
Composant	Partie réutilisable et centralisée de ses Logiciels.

Composant RAD	Composant mis en place facilement grâce à l'EDI.
Constructeur	Méthode paramétrée permettant de créer les variables d'un Objet et de l'initialiser. On déclare cette méthode particulière en commençant par "constructor".
Destructeur	Méthode paramétrée permettant de détruire les Objets d'un Objet. En PASCAL Objet on déclare cette méthode particulière en commençant par "destructor".
EDI ou IDE	Environnement de Développement Intégré, ou Integrated Development Environment en anglais. LAZARUS est un EDI. Il permet de créer un Logiciel grâce à un ensemble d'outils homogènes.
Embarqué ou Embedded	L'Embarqué c'est placer un Logiciel dans du matériel électronique.
Encapsulation	Déclarations permettant de réutiliser une partie d'un Objet.
Fonction	Procédure ou Méthode paramétrée retournant une variable.
Fork	Copie d'un projet Libre ou Open Source.
Framework	Savoir-faire Logiciel réutilisable permettant de réaliser un certain nombres de tâches.
Héritage	Procédés permettant à un Objet d'hériter d'un autre Objet.
Ingénierie Pilotée par Modèle	L'Ingénierie Pilotée par Modèle, ou Model Driven Engineering en anglais, est l'utilisation par un Ingénieur d'un Modèle afin de créer un Logiciel. Un Framework LAZARUS peut modéliser votre Logiciel afin de le créer.
Librairie	Ensemble de Composants et d'outils permettant des fonctionnalités ou un

	gain de temps.
Libre	Une création ou un Logiciel Libre c'est une création pour laquelle la licence Libre permet de : Étudier la Source. Utiliser la création librement. Modifier le Logiciel ou la création. Dupliquer tout ce qui a été fait. Diffuser commercialement ou pas ce qui a été créé. Une licence Libre peut être : Virale : La communauté grandit facilement, car on est obligé de diffuser la Source modifiée ou le Logiciel l'utilisant. La licence Libre virale la plus connue est la licence GPL qui est utilisée pour partie avec LAZARUS. Entièrement Libre. La création peut être modifiée commercialement sans avoir aucune contrainte de diffusion forcée de la Source. La licence BSD est une licence entièrement Libre. Du domaine public, car les auteurs ont abandonné les droits commerciaux ou l'œuvre a vu sa période de fin de droits d'auteur terminée. En général il est possible de s'approprier le projet du domaine public.
Logiciel	Ensemble de traitements permettant la résolution de tâches.
Méthode	Procédure ou fonction dans un Objet. Une méthode peut être encapsulée ou surchargée.
Objet (Analyse ou Programmation Objet)	L'analyse Objet est une analyse des systèmes d'informations proche de l'humain. Elle offre une architecture compréhensible par la machine. On représente toute entité abstraite ou concrète par un Objet qui dispose de propriétés Objet. Les quatre genres

	de propriétés d'un Objet sont l'Héritage, le Polymorphisme, l'Encapsulation, l'Abstraction. En respectant ces propriétés, on crée un programme d'Objets disposant de spécificités propres au langage utilisé.
Open Source	Projet à Sources partagées. Il peut être possible de participer au projet en fonction de la licence et de la disponibilité de l'entreprise.
Ordinateur	Boîte contenant une carte mère (le support), un processeur (l'unité de calculs), de la mémoire, d'éventuels périphériques.
Paquet	Projet LAZARUS permettant de réunir et d'installer des Composants.
Patch	Modification à ajouter à un exécutable. Le patch augmente ou diminue la taille de l'exécutable.
Pointeur	Un pointeur permet de stocker les variables pour les retrouver. Un pointeur est une adresse pointant vers un endroit de la mémoire.
Polymorphisme	Procédé permettant par différentes manières de réutiliser plusieurs Objets en même temps. Le Polymorphisme est résolu par le type Objet "interface" permettant de dénommer les méthodes communes de futurs Objets.
Procédure	Bout de Code commençant par la déclaration implémentée et paramétrable "procedure", délimitée par un "Begin" et un "End".
Procédural (Langage)	Les langages procéduraux sont des langages non Objets. Ils sont architecturés selon des unités de procédures et fonctions s'appelant entre elles. Le langage PASCAL était au début un langage uniquement

	procédural.
Propriétaire ou Privateur	Un Logiciel Propriétaire ne diffuse aucune source. Il est restreint. WINDOWS est Privateur dans le sens où c'est un Système d'Exploitation en location. En effet, vous êtes restreint au niveau du matériel par la nécessité de rester compatible avec les dernières Applications payantes.
RAD ou DRA	Rapid Application Development. Développement Rapide d'Applications grâce à un EDI ou une librairie.
Roadmap	Carte de route en mot à mot. Document ou outil de planification des évolutions.
Source	Recette qui permet de créer son Logiciel. Il est intéressant d'utiliser des projets Open Source ou Libres afin de ne pas travailler dans le flou.
Système d'Exploitation	Ou Environnement. Logiciel démarré à partir d'un disque dur ou d'une carte flash. Ce Logiciel permet d'utiliser son Ordinateur.
Trois tiers	Une application trois tiers passe par trois intermédiaires pour afficher des données. En serveur de données dialogue avec un serveur web ou trois tiers. Ce serveur dialogue respectivement avec un ou plusieurs navigateurs web, ou bien avec une application associée au serveur trois tiers.
Type	Définition d'une variable permettant de la manipuler facilement grâce au compilateur.
UML	Unified Modeling Language ou Langage Unifié de Modélisation. Boîte à outils de modèles d'analyse servant à analyser un projet basé sur

	la programmation Objet.
Variable	Partie de la mémoire, définie par un type, pouvant varier à l'exécution. Une variable stocke des chiffres, ou bien des lettres, adresses, Objets, etc.
Web	Réseau maillé pouvant contenir un nombre presque illimité d'ordinateurs, grâce au protocole IPV6. Les ordinateurs utilisent un navigateur Web, qui peut accéder aux applications participatives, dites Web 2.0.

ISBN 9791092732214

Éditions LIBERLOG
Éditeur n° 978-2-9531251

Dépôt Légal RENNES 2010

Imprimé en France en Juin 2018 par :
PRINT 24 FR